



# **OPAS MOBIILIPELIGRAFIIKAN TOTEUTTAMISEEN UNITY- PELINKEHITYSTYÖKALUN JA IOS- LAITTEIDEN VAATIMUKSET HUOMIOIDEN**

Juho Pietilä

Opinnäytetyö  
Toukokuu 2012  
Tietojenkäsittely  
Digimedia

## TIIVISTELMÄ

Tampereen ammattikorkeakoulu  
Tietojenkäsittely  
Digimedia

PIETILÄ, JUHO: Opas mobiilipeligrafiikan toteuttamiseen Unity-pelinkehitystyökalun ja iOS-laitteiden vaatimukset huomioiden

Opinnäytetyö 56 sivua, josta liitteitä 25 sivua  
Toukokuu 2012

---

Opinnäytteeni suunnittelu käynnistyi, kun Ifelse Media Oy halusi grafiikan olevan yhdenmukaisempaa mobiilipeliprojekteissa. Tästä syntyi idea mobiilipeligrafiikkaoppaasta, jonka tarkoituksena oli erityisesti huomioida Unity-ohjelman ja iOS-laitteiden vaatimukset sekä mahdollisuudet. Oppaan tarkoituksena oli luoda yritykselle raamit grafiikan tekemiseen mobiilipeliprojekteissa. Oppaan tarkoituksena oli myös vähentää ulkopuolisten työntekijöiden, kuten freelancergraafikoiden ja harjoittelijoiden, opastuksen tarvetta.

Työssä käsiteltiin grafiikan, peligrafiikan ja mobiilipeligrafiikan yleisluonteisia piirteitä sekä Unity-ohjelman ja iOS-laitteiden mobiilipeligrafiikalle luomia ehtoja. Opinnäytetyön raportissa on tarkoitus avata lukijalle mahdollisesti uusia käsitteitä, kuten piirto-komentoja ja niiden vaikutusta 3D-mallien toteuttamiseen. Opasosa puolestaan kuvaa grafiikan tekoa nimenomaan Ifelse Media Oy:ssä käytössä olevien standardien mukaisesti. Materiaali oppaaseen koottiin alan kirjallisuudesta, Internet-sivuilta ja henkilökohdasta osaamistani hyödyntämällä.

Oppaasta syntyi tiivis kokonaisuus, joka vastaa sille asetettuja tavoitteita. Kehitysideoita oppaalle on jo syntynyt, sillä kun yritys ryhtyy julkaisemaan pelejä Android-laitteille, opasta kehitetään tarpeiden mukaiseksi. Oppaan kehitykseen vaikuttaa myös käyttäjien antama palaute. Ammatillinen osaamiseni kasvoi myös paljon opinnäytetyön tekemisen ansiosta.

## ABSTRACT

Tampereen ammattikorkeakoulu  
Tampere University of Applied Sciences  
Degree Programme in Business Information Systems  
Option of Digimedia

PIETILÄ, JUHO: A Guide for Producing Mobile Game Graphics for Unity and iOS Devices

Bachelor's thesis 56 pages, appendices 25 pages  
May 2012

---

The planning of this Bachelor's thesis started when Ifelse Media Ltd wanted the graphics in their game projects to be more consistent. This led to an idea to create a guide which was designed specifically for the Unity programme and iOS devices' requirements and possibilities. The purpose of the guide was to create a framework for the company to make graphics in the game projects. Another aim was also to reduce the need for training of external employees, such as freelancer graphic designers and trainees.

The work covers general aspects of graphics, game graphics and mobile game graphics, as well as the requirements of the software and devices that are set up for graphics. The thesis explains concepts that might be new to reader, such as draw calls and their impact on the implementation of the 3D models. The guide describes how to make graphics with standards that are used in Ifelse Media Ltd. The data for the guide was collected from the literature, websites and by utilising the author's personal skills.

The guide became a compact entirety and the aims set were met. Furthermore, developing ideas for the guide have already been invented and when the company will start publishing games to Android devices, the guide will be revised to meet the new needs. The guide will also be amended on the basis of feedback from the users.

---

Key words: mobile game graphics, game graphics, Unity, iOS, guide

## SISÄLLYS

1	JOHDANTO.....	6
2	PELIGRAFIIKKA.....	8
2.1	2D-grafiikka.....	8
2.2	3D-grafiikka.....	8
2.3	Mobiilipeligrafiikka .....	9
3	IOS-LAITTEET .....	10
3.1	Yleistä .....	10
3.2	Laitteiden näytöt .....	10
3.2.1	Täyttönopeus .....	11
3.2.2	Läpinäkyvyyden käsittely .....	12
3.3	Muisti .....	12
3.4	Verteksien määrä .....	12
4	GRAFIIKKA UNITY-OHJELMASSA .....	14
4.1	Piirtokomennot.....	14
4.1.1	Dynaaminen piirtokomentojen yhdistäminen .....	14
4.1.2	Staattinen piirtokomentojen yhdistäminen.....	15
4.1.3	Staattinen yhdistäminen 3D-mallinnusohjelmassa .....	15
4.2	3D-mallien rakenne.....	15
4.3	3D-mallien materiaalit ja shaderit.....	17
4.3.1	Tekstuurit .....	18
4.3.2	Tekstuuriatlakset .....	18
4.3.3	Toistuvat tekstuurit .....	19
4.3.4	Normaalikartat.....	20
4.3.5	MIP-kartat .....	20
4.3.6	Tekstuurien pakkaus.....	21
4.3.7	UV-kartoitus.....	21
4.4	3D-mallien animaatio .....	23
4.5	3D-mallien vienti .....	24
4.6	Valaistus.....	24
4.7	Sovelluksen kuvakkeet .....	25
5	OPINNÄYTETYÖN TOTEUTUS .....	27
5.1	Opinnäytetyön tarkoitus ja prosessi.....	27
5.2	Opas .....	27
6	POHDINTA.....	29
	LÄHTEET.....	30
	LIITTEET .....	32

Liite 1. Mobiilipeligrafiikkaopas.....	32
--	----

## 1 JOHDANTO

Peliteollisuus on kehittynyt viime vuosina Suomessa paljon. Myös uusia peliyrityksiä perustetaan enemmän kuin ennen ja esimerkiksi Tampereella on syntynyt kymmeniä pelialan yrityksiä (Pirkanmaan ELY-keskus ja Tekes 2011). Yksi niistä on Ifelse Media Oy, jossa olen yhtenä osakkaana.

Ifelse Media Oy on pieni tamperelainen pelialan yritys, joka on perustettu syksyllä 2011. Yritys tekee päätoimenaan mobiilipelejä. Mobiilipelit tehdään tällä hetkellä iOS-laitteille, mutta tulevaisuudessa on tarkoitus laajentaa toimintaa myös Android-laitteille.

Tavoitteena työssä oli kehittää ja selvittää yritykselle standardeja mobiilipeligrafiikan tuottamiseen. Opinnäytetyön tavoitteena oli myös selvittää, kuinka valittuihin ratkaisuihin on päädytty, ja millaisia vaikutuksia niillä on mobiilipeliprojektissa. Henkilökohtaisena oppimistavoitteena oli oppia lisää peligrafiikan tekemisestä sekä siitä, miten kohdelaitteisto vaikuttaa grafiikan toteuttamiseen.

Opinnäytetyöni tarkoituksena oli tuottaa Ifelse Media Oy:lle opas mobiilipeligrafiikan tuottamiseen (LIITE 1). Opas kuvaa, kuinka mobiilipeligrafiikan tekemisessä täytyy huomioida erityisesti Unity-ohjelman ja iOS-laitteiden rajoitukset ja mahdollisuudet. Työn tarkoituksena oli siis toteuttaa opas, joka sisältää ohjeet mobiilipeligrafiikan toteuttamista varten. Oppaan avulla yritys saa selkeät raamit siitä, miten grafiikka pitäisi toteuttaa. Tarkoituksena on myös se, että yritys saa esimerkiksi freelancergraafikoilta teknisesti halutunlaista materiaalia. Oppaasta toivotaan olevan hyötyä myös alaa opiskeleville ja alasta kiinnostuneille henkilöille.

Työssäni käytän lähteinä kirjallisuutta sekä verkkoaineistoa. Opinnäytetyöni ensisijaisena lähteenä toimii Unity Technologies -yrityksen Internet-sivut tai tarkemmin sanotuna Unityn Internet-sivuilla sijaitseva manuaali. Työni toiseksi merkittävämpänä lähteenä voidaan pitää Applen iOS Dev Centerin Internet-sivuja. Molemmat lähteet ovat luotettavia ja ajantasaisia. Työssä käytetään muitakin verkkolähteitä. Tämä johtuu pääsääntöisesti siitä, että mobiilipeligrafiikasta on olemassa suhteellisen vähän kirjallisuutta. Verkkolähteissä olen pyrkinyt valitsemaan uusia julkaisuja, sillä mobiilipeligrafiikassa huomioitavat asiat saattavat muuttua melko nopeasti jo lyhyessäkin ajassa.

Tässä työssä termillä iOS-laitteet tarkoitetaan seuraavia Applen tuotteita: iPhone 3gs, iPhone 4, iPhone 4S, iPad, iPad 2 ja kolmannen sukupolven iPad. Tuotoksessa käytetään myös sanaa mobiililaite, jolla viitataan puhelinlaitteisiin eikä kaikkiin kannettaviin laitteisiin. Työ rajattiin yrityksen käytössä oleviin ohjelmistoihin ja kohdelaitteisiin, joille peliprojektit toteutetaan. Myös 2D-grafiikan animaatio rajattiin pois työstäni.

## 2 PELIGRAFIikka

### 2.1 2D-grafiikka

2D-grafiikalla tarkoitetaan kaksiulotteisia kuvia tai tekstejä sekä niiden valmistukseen käytettyjä tekniikoita. 2D-grafiikka sisältää siis kaksi ulottuvuutta, jotka ovat pituus ja leveys. 2D-grafiikka voidaan toteuttaa vektorigrafiikkana tai bittikarttagrafiikkana (TechTerms.com 2009). Suurin ero niiden välillä on, että vektorigrafiikkaa voidaan skaalata ilman, että se hajoaa tai muuttuu rakeiseksi.

Ensimmäinen reaaliaikaista grafiikkaa sisältävä tietokonepeli Spacewar! julkaistiin vuonna 1961 ja sen toteutuksessa käytettiin vektorigrafiikkaa (Puhakka 2008, 25). Kun pelit yleistyivät, ruvettiin niissä pääsääntöisesti käyttämään bittikarttagrafiikkaa. Bittikarttagrafiikka koostuu pienistä neliöistä, joita kutsutaan pikseleiksi. Kuvan jokaisessa pikselissä on tieto sen punaisesta, vihreästä ja sinisestä väriarvosta (Duggan 2008, 30).

Vaikka peli toteutettaisiinkin 3D-grafiikalla, on 2D-grafiikalla vielä todella suuri merkitys niissä. Tämä johtuu siitä, että 3D-mallit ovat peleissä yleensä melko yksinkertaisia ja 2D-grafiikan avulla voidaan luoda objekteista monimutkaisemman näköisiä. (Lehtovirta & Nuutinen 2000, 147.)

### 2.2 3D-grafiikka

Peliteollisuus on noussut tuloissa jo jonkin aikaa sitten viihdeteollisuudessa filmitelisuuden ohi. Tärkeimpänä syynä voidaan pitää kolmiulotteisen tietokonegrafiikan yleistymistä peleissä (Lehtovirta & Nuutinen 2000, 139). Vuonna 1993 julkaistu Doom-peli avasi oven 3D-grafiikan käyttämiseen tietokonepelissä (Puhakka 2008, 27). Tietotekniikan ja 3D-tekniikan kehittyminen on mahdollistanut sen, että pelit ovat tulleet kokoajan realistisemmiksi (Lehtovirta & Nuutinen 2000, 139).

3D-mallinnus on tehtävä peleihin tehokkaasti, sillä pelin grafiikka renderöidään eli piirretään reaaliaikaisesti laitteen näytölle. Renderöimisellä tarkoitetaan siis kolmiulotteisen mallin projisointia kaksiulotteiselle pinnalle. Kun 3D-elementtejä mallinnetaan, täytyy



muistaa pitää polygonimäärä alhaisena. Monimutkaisissa malleissa tämä vaatii taitoa, sillä mallin muoto joudutaan tekemään paljon yksinkertaisemmin. Polygonien maksimimäärä riippuu kuitenkin täysin kohdelaitteen tehosta. (Lehtovirta & Nuutinen 2000, 144–145; Puhakka 2008, 163.)

### **2.3 Mobiilipeligrafiikka**

Ensimmäinen matkapuhelinpeli ilmestyi vuonna 1997, kun Nokia julkaisi tietyissä matkapuhelimissaan matopelin (Steinbock 2005, 151). Vaikka matkapuhelinpelien kehitys yksivärisistä peleistä teräväpiirto peleihin ei ole ollut kovin suoranaista, on se muuttanut käsitystä siitä, mitä matkapuhelimilla voidaan nykypäivänä tehdä (Corbo 2011).

Apple julkaisi vuonna 2007 ensimmäisen sukupolven iPhone-laitteen, jonka ansiosta matkapuhelinpelit saivat aivan uudenlaisen käsityksen. Ensimmäisen iPhone-laitteen julkaisun jälkeen, kaikki iPhone-laitteet ja muut älypuhelimet ovat olleet hyviä alustoja pelaamiseen. Älypuhelimet omaavat tarvittavan teknologian useiden erityyppisten pelien toteuttamiseen, kuten teräväpiirtonäytöt ja kosketusnäytöt. Nopeat yhteydet, jotka mahdollistavat tiedostokooltaan suurten pelien lataamisen laitteille, ovat tuoneet 3D-grafiikkaa sisältävät pelit älypuhelimille. (Corbo 2011.)

### 3 iOS-LAITTEET

#### 3.1 Yleistä

Pelinkehityksessä on tärkeää valita laitteet, joilla pelin on tarkoitus toimia. Hyvä puoli pelinkehityksessä iOS-laitteille on se, että laitteita on suhteellisen rajattu määrä, toisin kuin monilla muilla mobiilialustoilla. Usein joudutaan tekemään valinta, toimiiko peli kaikilla laitteilla vai pelkästään niillä, jotka mahdollistavat pelin toimimisen täydellä teholla ja säilyttämään samalla grafiikka-resurssien laadun (McDermott 2010, 24).

Ennen peligrafiikan tuottamista on hyvä ymmärtää edes hieman, kuinka kohdelaitteet ja pelinkehitystyökalu toimii. Kaikilla eri iOS-laitteilla on omat kompastuskivensä. Sovellus saattaa toimia moitteettomasti jollakin laitteella, kun taas toisella laitteella se ei välttämättä toimi yhtä sulavasti (McDermott 2010, 1).

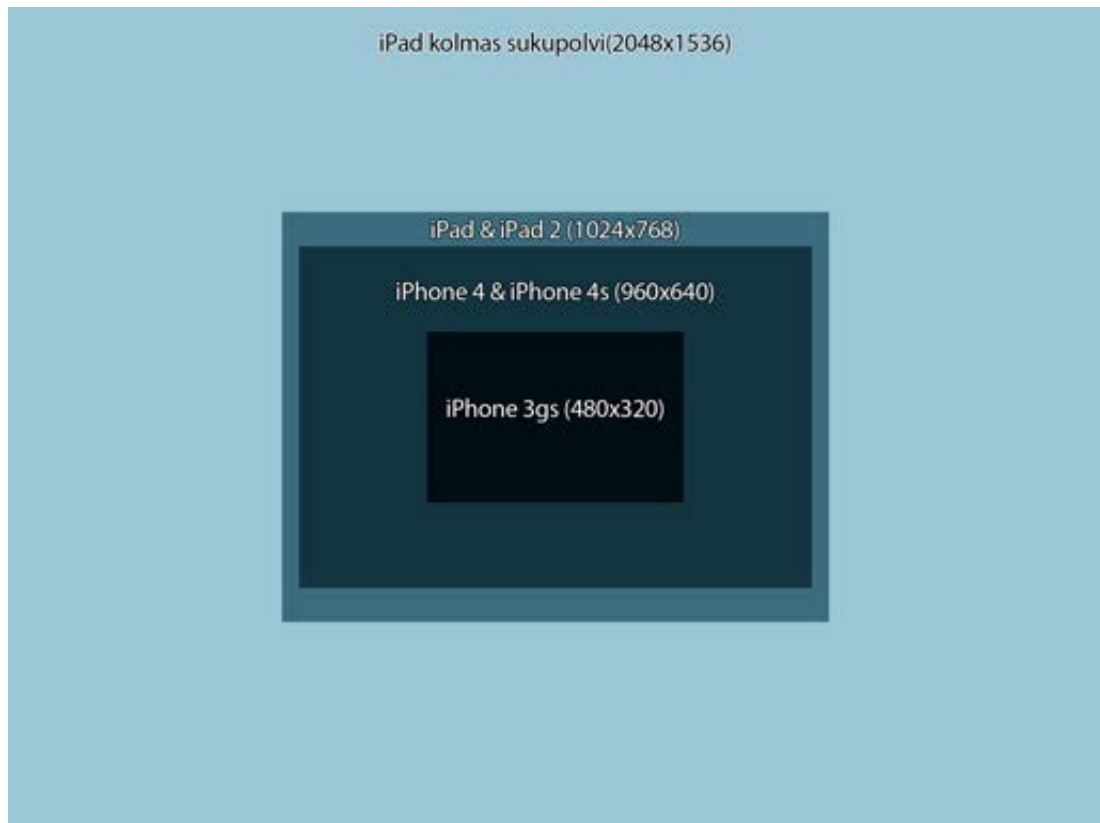
#### 3.2 Laitteiden näytöt

iOS-laitteiden näyttöjen resoluutiot eroavat toisistaan selkeästi (taulukko 1). Peligrafiikka tehdessä tulee huomioida kohdelaitteiden resoluutiot, sillä jos peligrafiikka toteutetaan esimerkiksi iPhone 3gs -laitteelle ja skaalataan iPad-laitteelle, on lopputulos kameran näköinen (McDermott 2010, 8).

TAULUKKO 1. iOS-laitteiden näyttöjen resoluutiot (Apple Inc 2012a; Apple Inc 2012b; Apple Inc 2010)

Laite	Resoluutio	Pikseliä tuumalla
iPhone 3gs	480x320	163 ppi
iPhone 4, iPhone 4S	960x640	326 ppi
iPad, iPad 2	1024x768	132 ppi
iPad kolmas sukupolvi	2048x1536	264 ppi

Kuvan 1 avulla on tarkoitus havainnollistaa, kuinka suuri ero iOS-laitteiden näyttöjen resoluutiolla on. Kuva on pienennetty viisi kertaa todellista kuvaa pienemmäksi.



KUVA 1. iOS-laitteiden näyttöjen resoluutiot

### 3.2.1 Täyttönopeus

Täyttönopeus on pikselien lukumäärä, jonka grafiikkaprosessori kykenee renderöimään laitteen näytölle sekunnissa. iOS-laitteista iPhone 3gs, iPhone 4 sekä iPad käyttävät samaa SGX 535 -grafiikkaprosessoria, vaikka iPhone 4 - ja iPad-laitteiden näytöt ovatkin paljon suurempia resoluutioltaan. Tämä voi muodostua ongelmaksi, sillä grafiikkaprosessori joutuu tekemään 4–5 kertaa enemmän töitä sovelluksen piirtämisessä iPhone 4 - ja iPad-laitteilla verrattuna iPhone 3gs -laitteeseen. Erityisesti läpinäkyvät pinnat ovat haasteellisia, sillä ne saattavat dramaattisesti tiputtaa ruudunpäivitysnopeutta iPhone 4 - ja iPad-laitteilla, vaikka sovellus toimisikin sulavasti iPhone 3gs -laitteilla. Grafiikkaprosessorin täyttönopeutta voidaankin siis pitää rajoitettuna iPhone 4 - ja iPad-laitteilla ja sitä voidaan pitää näiden laitteiden kompastuskivenä. (McDermott 2010, 8–9.)

### 3.2.2 Läpinäkyvyyden käsittely

Unity-ohjelmassa on kaksi eri tapaa käsitellä läpinäkyvyyttä. Ensimmäinen tapa on alfa-blending, joka on huomattavasti kevyempi vaihtoehto. Alfa-blending ei vaadi suoritukseen lisämuistia, koska väriarvot luetaan kuvaruutupuskurista. Toinen vaihtoehto on alfa-testing, ja se on huomattavasti järeämpi vaihtoehto, koska läpinäkyvyysarvoa verrataan kiinteään arvoon. (McDermott 2010, 7, 11.)

### 3.3 Muisti

Prossessorin ja grafiikkaprosessorin suoritusnopeus eivät ole ainoat tekijät sovelluksen sulavan toiminnan takaamiseksi iOS-laitteilla (McDermott 2010, 7). RAM-muistin määrä iOS-laitteilla on laitekohtaista ja se vaihtelee 256 MB aina 1 GB asti (taulukko 2). Tekstuurit kuluttavat yleensä eniten RAM-muistia iOS-laitteilla. Varsinkin suuret tekstuurit saattavat muodostua ongelmiksi, jos RAM-muistia ei ole paljon laitteen käytettävissä. Tekstuureja kannattaakin optimoida mahdollisimman tehokkaasti pakkaamalla niitä, etteivät ne kuluttaisi niin paljoa RAM-muistia (McDermott 2010, 11).

TAULUKKO 2. iOS-laitteiden muistin määrät (Kim 2010; Dilger 2011; White 2011; Slivka 2012)

Laite	Muisti
iPhone 3gs	256 MB
iPhone 4, iPhone 4S	512 MB
iPad	256 MB
iPad 2	512 MB
iPad kolmas sukupolvi	1 GB

### 3.4 Verteksien määrä

Verteksidata on iOS-laitteilla tärkeämpää kuin kolmiopolygonien lukumäärä. Verteksidata sisältää verteksit, jotka muodostavat 3D-mallin ja jokaisen verteksin tiedot. Tiedot sisältävät normaalit ja UV-datan. Verteksidatalla tarkoitetaan siis piirrettävän verteksin koordinaatteja, vektorin normaalia, tekstuurin koordinaatteja sekä väriarvoa. Kolmiopo-

lygonien lukumäärä on puolestaan määrä, joka muodostuu, kun kaikki monikulmioverkon kolmiopolygonit lasketaan yhteen. (McDermott 2010, 3–4.)

Kaikkien verteksin tiedot kopioidaan ja siirretään piirtokomentojen avulla jokaiseen pelisilmukan kehykseen. Verteksidata muutetaan tai siirretään 3D-avatuuteen prosessorilla. Aikaansaatu muunnos siirretään sisäiseen verteksipuskuriin. Puskuri on iso lista tai säiliö, joka sisältää kaiken verteksidatan. Verteksidatan kopioiminen vie hieman aikaa ja tämän vuoksi se tuhlaa muistia. Sen vuoksi verteksidata on syytä pitää mahdollisimman alhaisena. (McDermott 2010, 3–4.)

Pelisilmukan jokaisessa kuvaruudussa on aina tietty määrä verteksejä näkymässä. Verteksien kokonaismäärä olisi syytä pitää alle 40 000:n, kun peli aiotaan toteuttaa iPhone 3gs - tai uudemmille laitteille (Unity Technologies 2011).

## 4 GRAFIikka UNITY-OHJELMASSA

### 4.1 Piirtokomennot

Piirtokomennot voidaan mieltää Unity-ohjelman lähettämiksi pyynnöiksi kohdelaitteelle piirtää objekti scene-näkymään. Piirtokomento syntyy aina silloin, kun prosessori lähettää grafiikkaprosessorille käskyn renderöidä jotakin. Grafiikkaprosessori pystyy käsittelemään dataa erittäin nopeasti, jos tieto on lähetetty suurina määrinä, mutta se ei pysty käsittelemään tietoa kovin tehokkaasti per kuvaruutu. Tämän vuoksi liiallisten piirtokomentojen määrä saattaa aiheuttaa kompastuskiven laitteen suorituskyvylle. Onkin suositeltavaa, että grafiikkaprosessorille lähetetään kerralla paljon tietoa prosessoitavaksi. Unity-ohjelmassa on mahdollista vähentää piirtokomentojen määrää yhdistämällä niitä. Piirtokomentoja voidaan yhdistää kahdella eri tavalla, dynaamisella ja staattisella. (McDermott 2010, 3, 5.)

#### 4.1.1 Dynaaminen piirtokomentojen yhdistäminen

Objektien pitää täyttää tietyt kriteerit, jotta ne voidaan yhdistää dynaamisesti. Yhdistämisen mahdollistamiseksi objektien täytyy jakaa sama materiaali, eikä objekteissa saa olla liikaa verteksejä. Jos nämä kriteerit täyttyvät, yhdistää Unity-ohjelma objektit samaan piirtokomentoon automaattisesti. Piirtokomentojen yhdistäminen ei siis aiheuta käyttäjälle ylimääräistä työtä, jos objektit ovat toteutettu oikealla tavalla (Unity Technologies 2012a).

Objektin verteksien maksimimäärä riippuu verteksiattribuuttien määrästä. Verteksiattribuutilla tarkoitetaan jotakin verteksin ominaisuutta, kuten esimerkiksi sijaintia, normaalia tai UV-karttaa. Alle 900 verteksin objektit voidaan yhdistää, jos vertekseillä on vain yksi attribuutti. Jos vertekseillä on enemmän attribuutteja kuin yksi, voidaan objektin verteksien maksimimäärä laskea jakamalla 900 attribuuttien määrällä. Esimerkiksi, jos objektin vertekseillä on kolme attribuuttia, saadaan maksimimääräksi 300 verteksiä (Unity Technologies 2012a).

#### **4.1.2 Staattinen piirtokomentojen yhdistäminen**

Staattinen yhdistäminen on toinen tapa, jolla voidaan vähentää piirtokomentoja Unity-ohjelmassa. Se toimii melkein samalla tavalla kuin dynaaminen yhdistäminen. Suurin ero dynaamisessa ja staattisessa yhdistämisessä on se, että staattisesti yhdistetyt objektit eivät voi liikkua scene-näkymässä sovelluksen toiminnan aikana. Staattisen yhdistämisen mahdollistamiseksi täytyy halutut objektit merkitä Unity-ohjelmassa staattisiksi ja merkittyjen objektien pitää jakaa sama materiaali. Staattisessa yhdistämisessä ei ole verteksien maksimimäärärajaa niin kuin dynaamisessa yhdistämisessä. Staattinen yhdistäminen sopii parhaiten taustaobjekteille, jotka eivät liiku sovelluksen toiminnan aikana. Staattinen piirtokomentojen yhdistäminen on mahdollista vain Unity iOS Advanced -ohjelmistolisenssillä. (McDermott 2010, 6; Unity Technologies 2012a.)

#### **4.1.3 Staattinen yhdistäminen 3D-mallinnusohjelmassa**

Objekteja voidaan yhdistää staattisesti myös 3D-mallinnusohjelmassa. Tämä tarkoittaa käytännössä sitä, että useat objektit yhdistetään yhdeksi objektiksi. Objektien yhdistämisessä on hyvä ottaa huomioon, että yhdistettävien 3D-mallien täytyy esiintyä pelin aikana yhdessä. Objektiryppäälle on suositeltavaa tehdä yhteinen tekstuuri sen sijaan, että jokaiselle yhdistettävälle objektille tehtäisiin oma tekstuuri. Mallinnusohjelmassa tapahtuva yhdistäminen sopii parhaiten taustaobjekteille (McDermott 2010, 82–84).

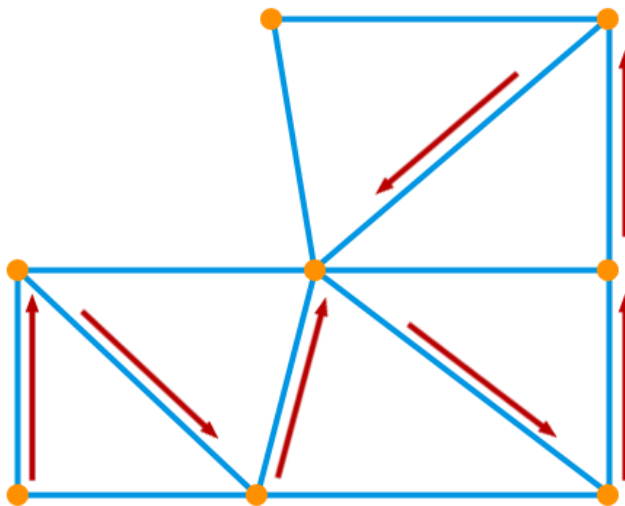
### **4.2 3D-mallien rakenne**

Monikulmioverkkoja käytetään usein esittämään 3D-mallien pintoja. 3D-malli muodostuu siis usein monikulmioverkosta, joka on rakennettu pinnoista, vertekseistä ja sivuista. Pinta koostuu vertekseistä eli kulmapisteistä. Monikulmioverkossa viereiset pinnat kohtaavat toisensa jakamalla vähintään yhden verteksin. Monikulmioverkon verteksimäärä vaikuttaa siihen, kuinka paljon yksityiskohtia on mahdollista tehdä monikulmioverkkoon. (Lehtovirta & Nuutinen 2000, 21.)

Unity-ohjelmassa ei ole mahdollista luoda monikulmioverkkoja, mutta se tukee todella hyvin useita 3D-mallinnusohjelmia (Unity Technologies 2012b). Objektien verteksimäärä on todella tärkeä, jotta esimerkiksi piirtokomentojen yhdistäminen on mahdollista.

Objekteja ei saa skaalata Unity-ohjelmassa, koska ohjelma tekee skaalatusta objektista kopion. Kopioiden tekeminen kuluttaa muisti resursseja (McDermott 2010, 25). Verteksimäärä on todella tärkeä iOS-laitteille suunnatuissa peleissä. Sen vuoksi on hyvä tietää, että 3D-mallinnusohjelman ilmoittama verteksimäärä ei ole objektin todellisten verteksimäärä. Ainoa mahdollinen tapa selvittää todellinen verteksimäärä on tuoda objekti Unity-ohjelmaan. Useat eri tekijät saattavat vaikuttaa siihen, että objektin verteksimäärä kasvaa, kun se renderöidään grafiikkaprosessorilla. (McDermott 2010, 29.)

Unity-ohjelma käyttää kolmiopolygoniketjuja, jotta se voi näyttää monikulmioverkot mahdollisimman tehokkaasti (kuva 2). Kolmiopolygoniketjut ovat yhteenliitettyjä kolmiopolygoneja, jotka mahdollistavat nopeamman renderöinnin ja tehokkaamman muistin käytön (McDermott 2010, 30).

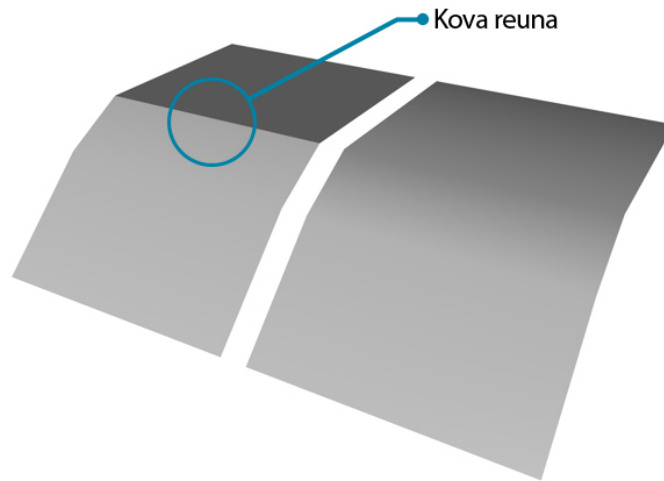


KUVA 2. Kolmioketju (mukaillen McDermott 2010, 30)

Kulmanpyöristyksen avulla määritetään suurin poikkeama yhdistettyjen polygonien välillä. Mikäli polygonien välinen kulma on suurempi kuin kulmanpyöristys, ei pyöristystä toteuteta.



Verteksimäärä kasvaa, kun grafiikkaprosessorin täytyy renderöidä kovia reunoja (kuva 3). Kovat reunat syntyvät, kun yhdistetyt polygonit muodostavat kulman, joka on suurempi kuin kulmanpyöreys. Grafiikkaprosessorin pitää jakaa verteksit, jotta kova reuna saadaan aikaiseksi, ja tämä operaatio lisää verteksien kokonaismäärää. (McDermott 2010, 30.)



KUVA 3. Kova reuna ja kulmanpyöristys.

### 4.3 3D-mallien materiaalit ja shaderit

*"Materiaalit ovat oleellinen osa 3D-visualisointia. Ilman oikeita materiaaleja hienoin-kin 3D-malli näyttää samalta kuin vastarakennettu, aito muovista tehty maalaamaton puutalo"* (Lehtovirta & Nuutinen 2000, 30). Materiaalien avulla voidaan asettaa 3D-mallille ulkoasu ja ne ovatkin tunnettu käsite 3D-mallinnusohjelmissa sekä Unity-ohjelmassa. Materiaalit voivat olla tavallisista perusväreistä aina heijastaviin kuvapohjaisiin pitoihin asti (Goldstone 2009, 27).

Materiaaleilla ja shadereilla on läheinen yhteys Unity-ohjelmassa. Shaderit sisältävät koodia, joka määrittelee, millaisia ominaisuuksia ja grafiikkaresursseja käytetään. Kun taas materiaalit mahdollistavat grafiikkaresurssien asettamisen ja ominaisuuksien muokkaamisen. (Unity Technologies 2010a.)

Unity-ohjelmassa materiaalien käyttäminen on hyvin helppoa. Kun 3D-malli tuodaan Unity-ohjelmaan, luo pelimoottori materiaalin automaattisesti. Myös uusien materiaalien luonti Unity-ohjelmassa on helppoa ja niihin voidaan soveltaa vaivattomasti tekstuurreja (Goldstone 2009, 27).

#### **4.3.1 Tekstuurit**

Tekstuurit ovat joko valokuvia tai tietokoneohjelmalla luotuja kuvia. Tekstuurien avulla voidaan määrittää materiaalin tietty attribuutti. Niiden avulla voidaan määrittää esimerkiksi materiaalin väri. (Watkins 2011, 63.)

Tekstuureja tehdessä on aina syytä tehdä se mahdollisimman isoksi ja skaalata tekstuuria tarvittaessa pienemmäksi. Tämä mahdollistaa sen, että tekstuureissa voidaan säilyttää korkea laatu. Tekstuurit on myös syytä muistaa tehdä leveydeltään ja korkeudeltaan kahden potenssiin, jotta ne on mahdollista pakata tarvittaessa PVRTC-formaatilla. Tekstuurien ei tarvitse olla neliönmallisia. Tekstuurit voivat olla esimerkiksi seuraavan kokoisia 64x64, 256x128, 512x512 tai vaikkapa 1024x512. Lopullinen tekstuurikoko määräytyy useasta eri asiasta, kuten kohdelaitteen näytön resoluutiosta ja tekstuuri muistista. Jos objekti näkyy näytöllä koko pelin toiminnan aikana pienenä, ei siinä ole syytä säilyttää niin suurta tekstuuria. Jos objektin esiintyy useamman kokoisena sovelluksessa, kannattaa harkita Mip-karttojen käyttämistä. (McDermott 2010, 51.)

#### **4.3.2 Tekstuuriatlakset**

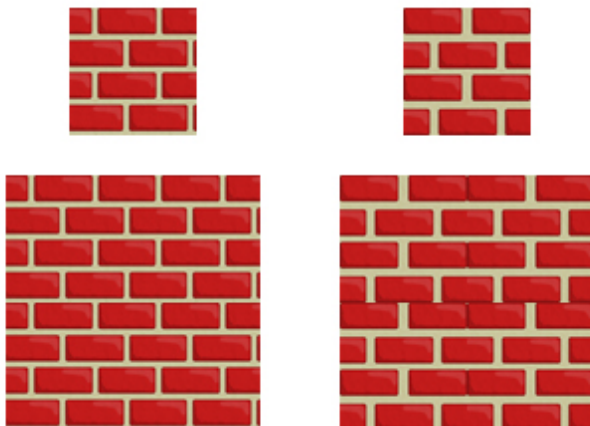
Tekstuuriatlas on yksi suuri kuva, joka sisältää usean erillisen objektin tekstuurit, ja niiden käyttäminen projektissa on loistava tapa optimoida tekstuureja (kuva 4). Niiden käyttäminen vähentää myös tekstuurien ja materiaalien määrää projektissa. Tekstuuriatlakset on syytä suunnitella ja toteuttaa huolella, sillä tyhjän tilan jättäminen tekstuuuriatlakseen tarkoittaa menetettyä mahdollisuutta grafiikan yksityiskohtien lisäämiseen (McDermott 2010, 76).



KUVA 4. Tekstuuriatlas

#### 4.3.3 Toistuvat tekstuurit

Toistuvien tekstuurien käyttöä kannattaa suosia, koska niitä käyttämällä saadaan tekstuurikokoja pienemmiksi. Kun luodaan toistuvia tekstuureja, pitää ne testata huolella, jottei tekstuurin toistumisesta synny saumoja (kuva 5). Jos tekstuuria toistettaessa näkyy saumoja, rikkovat saumat illuusion jatkuvasta rikkoutumattomasta pinnasta (Goldstone 2009, 65).



KUVA 5. Toistuva teksturi

#### 4.3.4 Normaalikartat

Normaalikarttojen avulla saadaan objekteihin lisättyä tarkkuutta sekä yksityiskohtia. Niitä käytetään 3D-malleissa samalla tavalla kuin tekstuureja, mutta niitä ei saa kohdella samalla tavalla kuin tekstuureja. Tämä tarkoittaa sitä, että normaalikarttojen valmistaminen eroaa huomattavasti perinteisestä 3D-mallin ja tekstuurin valmistamisesta. Normaalikarttojen tekemiseen on olemassa useita tekniikoita. Pelikehityksessä käytetään useimmiten niin kutsuttua tangent-based normal map -tekniikkaa. Tälle tekniikalle on tunnusomaista sininen väri, ja se onkin helppo tunnistaa siitä. Haittapuolena normaalikartoissa on se, että niiden tekeminen lisää grafiikan tuottamiseen tarvittavaa aikaa (Gahan 2009, 71).

Normaalikarttoja on olemassa kahta eri päätyyppiä runsasdetaljinen ja niukkadetaljinen. Runsasdetaljinen normaalikartta sisältää paljon yksityiskohtia, kuten naarmuja ja kuoppia. Ne luodaan yleensä valokuvista automaattisesti, koska tällaisten yksityiskohtien mallintaminen olisi liian työlästä ja se veisi liian paljon aikaa. Niukkadetaljisella normaalikartalla pyritään luomaan objektista tarkemman ja monimutkaisemman näköinen. Molempia tekniikoita yhdistäen saadaan luotua monimutkaisempia normaalikarttoja. Ennen kuin normaalikarttoja ryhdytään tekemään, kannattaa miettiä, onko niistä todellista hyötyä, sillä niiden tekeminen vie runsaasti aikaa. (Gahan 2009, 74–75.)

#### 4.3.5 MIP-kartat

MIP-kartta on optimoitu tekstuuri, joka mukailee alkuperäistä tekstuuria ja sen avulla voidaan nopeuttaa renderöintiä. MIP-kartta sisältää eri kokoisia versioita alkuperäisestä tekstuurista. Objektin etäisyys kamerasta vaikuttaa siihen, minkä kokoista tekstuuria käytetään. Eli mitä kauempana objekti on kamerasta, sitä pienempää kuvaa käytetään MIP-kartasta. MIP-karttojen avulla voidaan optimoida tehokkaasti tekstuureja. MIP-karttoja ei kannata käyttää, jos ennalta tiedetään, että objektia ei tulla näyttämään pienempänä (McDermott 2010, 55).

#### 4.3.6 Tekstuurien pakkaus

iOS-laiteet käyttävät pakkauksessa PVRTC-formaattia ja sen avulla on mahdollista pakata tekstuuri 2 tai 4 bittiä per pikseli. Pakkaamisen avulla saadaan vähennettyä muistin käyttöä. Pakkaaminen on siksi tärkeää, että iPhone-laite jakaa muistin prosessorin ja grafiikkaprosessorin välillä. Pakkaamattomat tekstuurit kuluttavat paljon RAM-muistia, jota on jo muutenkin rajallisesti laitteilla käytössä. Jos tekstuurit vievät liikaa muistia, voi pahimmassa tapauksessa laite kaatua sovelluksen käynnissäoloaikana. Suurien tekstuurien lataaminen RAM-muistiin vie myös enemmän aikaa. (McDermott 2010, 14, 51.)

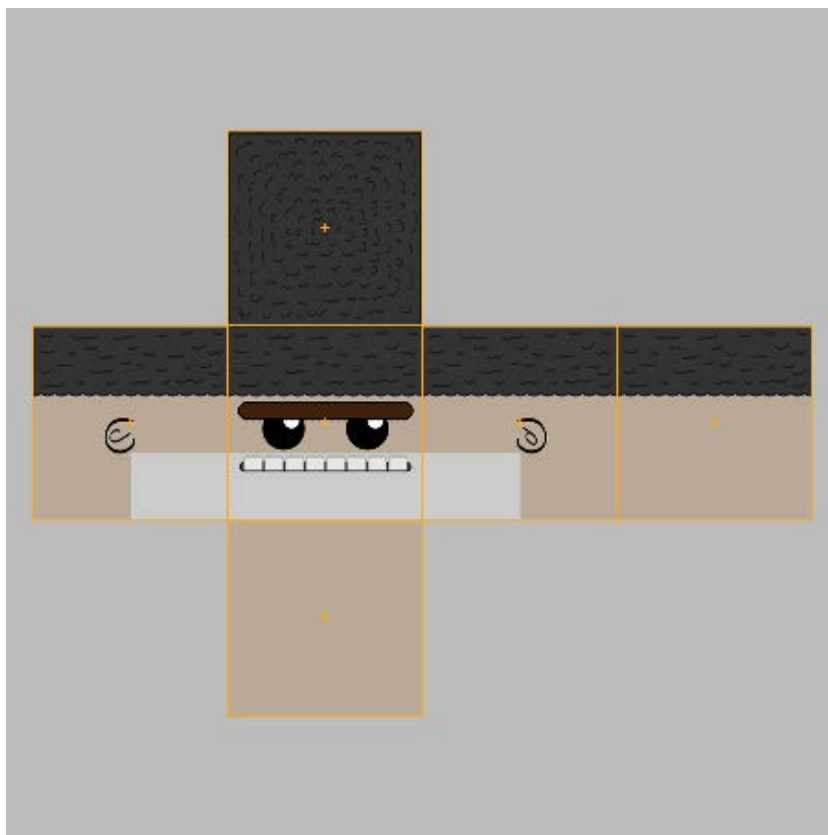
Tekstuurien täytyy olla neliöitä tai kahden potenssissa. PVRTC on ainoa pakkausmuoto, joka voi iOS-laitteilla jäädä muistiin pakattuna ja joka voidaan dekodata pakatussa muodossa grafiikkaprosessorilla. Tämä on tärkeää sen vuoksi, että iOS-laitteiden muistinkaistanleveys on vähäistä. Tekstuurit ovat usein syyllisiä siihen, että iOS-laitteiden muisti loppuu (McDermott 2010, 53).

#### 4.3.7 UV-kartoitus

UV tarkoittaa koordinaatteja ja ne on kehitetty ratkaisuksi yhteen monimutkaisimmista ongelmista. UV-koordinaattien avulla pystytään laittamaan kaksiulotteinen tekstuuri 3D-muodon ympärille. UV-tilojen muokkaaminen on yksi pelinkehityksen kannalta tärkeimpiä asioita saavuttaa visuaalisesti näyttävä tuote.

UV-kartoitus on prosessi, jossa kolmiulotteisen objektin muodot avataan kaksiulotteiseen esitysmuotoon. Tämä mahdollistaa sen, että objekteihin voidaan maalata tekstuuri. UV-kartoituksessa muodostuneet UV-koordinaatit ovat paikkoja objektin pinnalla, joihin on kiinnitetty tekstuuri. Kun koordinaatit on määritetty pinnoille, ei haittaa vaikka pinta taipuu tai vääristyy, sillä tekstuuri taipuu ja vääristyy pintojen mukana. UV-kartoitus auttaa siinä, että tekstuuri näyttää oikealta pinnalta. Huonosti toteutettu UV-kartta näkyy tekstuurissa venymisenä ja puristumisena. Hyvin toteutetulla UV-kartalla pystytään kätkemään melko yksinkertainenkin objekti niin, että tekstuuri tuottaa objektille uskottavan pinnan. Peleissä UV-kartat ovat todella tärkeässä roolissa, sillä tekstuurit luovat suuren osan pelien visuaalisuudesta. (Watkins 2011, 38.)

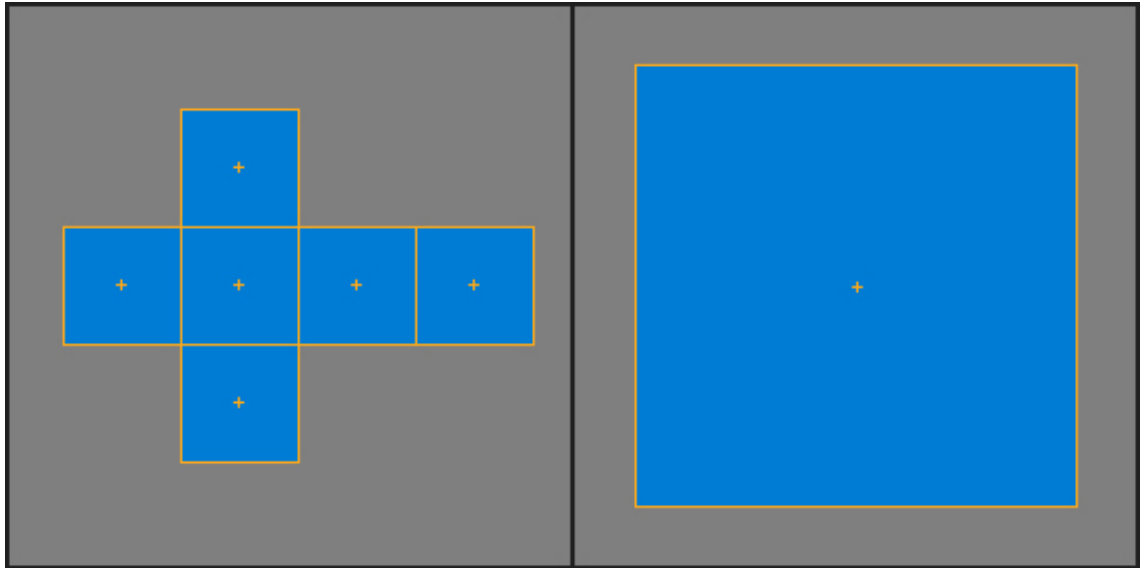
Kuvassa 6 esiintyy kuution muotoisen hahmon UV-kartta. UV-kartta näkyy kuvassa oranssin sävyisinä viivoina. Kuvassa näkyy myös kuvankäsittelyohjelmalla tehty tekstuuri.



KUVA 6. UV-kartta

UV-kartoissa muodostuu yleensä UV-sarjoja, jotka ovat erillään toisistaan. UV-sarjat muodostavat reuna-alueita, jotka sisältävät yhteisiä verteksejä. Vaikka verteksien yhteys onkin katkennut UV-kartassa, ovat ne yhä kytkettyinä toisiinsa monikulmioverkossa. Kun grafiikkaprosessori käsittelee UV-kartan reunaa, täytyy verteksit jakaa reunan perusteella ja tämä lisää verteksien kokonaismäärää. Tämän vuoksi on syytä pitää reunojen määrä minimissä, kun UV-karttaa luodaan objektille. (McDermott 2010, 30–31.)

Kuvassa 7 on tehty kuutiolle kahdella eri tavalla UV-kartta. Vasemmanpuoleisessa UV-kartassa muodostuu ulkolaidoista reunat, jotka lisäävät verteksien määrää. Oikeanpuoleisessa UV-kartassa verteksejä ei tarvitse jakaa reunan perusteella.

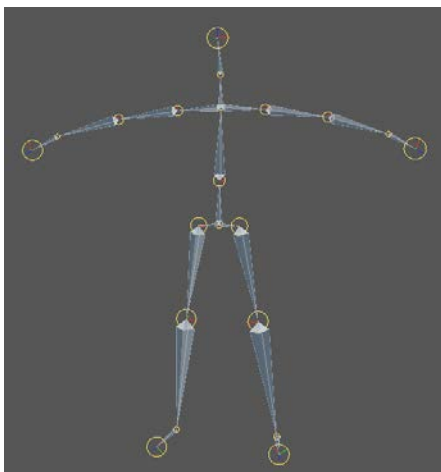


KUVA 7. UV-kartat

#### 4.4 3D-mallien animaatio

Usein 3D-mallit ovat muutakin kuin staattisia objekteja, ja sen vuoksi niissä halutaan esittää liikettä tai jotain monimutkaisempaa toimintaa. Tärkeimpänä voidaan pitää hahmon liikkumista esimerkiksi ihmisen kävelemistä, hyppimistä tai jotain muuta liikettä. (Puhakka 2008, 431.)

Luuston avulla voidaan päästään hyvään lopputulokseen hahmon animoinnissa. Luurankomallin avulla voidaan ohjata ja rajoittaa hahmon liikkeitä samalla tavalla niin kuin oikean ihmisen luusto toimii. Luurankomalli koostuu nivelistä ja nivelvarsista, joita voidaan kutsua myös luiksi (kuva 8).



KUVA 8. Luurankomalli

Useasti hahmon luurankomallin peruspisteeksi valitaan lantio. Luurankomallia voidaan liikuttaa nivelten asemaa muuttamalla. Luurankomallin avulla voidaan toteuttaa avain-asemiin perustuvaa animaatiota. (Puhakka 2008, 431–432.)

Luurankomalli täytyy pinnoittaa iholla, jotta hahmo olisi luonnollisen näköinen. Ihona luurankomalleissa toimii yleensä monikulmioverkko. Monikulmioverkon verteksit sidotaan luurankomalliin, ja jokainen yksittäinen verteksi sidotaan yhteen tai useampaan luuhun (Puhakka 2008, 432).

#### 4.5 3D-mallien vienti

Unity-ohjelma tukee useita tunnettuja 3D-mallinnusohjelmia ja niiden eri tallennusformaatteja (taulukko 3). Ohjelma tukee myös muista 3D-ohjelmista tuotuja 3D-malleja, jos niistä voidaan tallentaa objekti johonkin seuraavista tallennusmuodoista: .fbx, .dae, .3DS, .dxd tai .obj. (Unity Technologies 2010b.)

TAULUKKO 3. Unity-ohjelman tukemat ohjelmakohtaiset tallennusmuodot

Ohjelman nimi	Tuettu tallennusmuoto
Cinema 4D	.c4d
Blender	.blend
Maya	.mb, .ma
3ds Max	.max
Cheetah 3D	.jas
Modo	.lxo
Lightwave	.fbx

#### 4.6 Valaistus

Sovelluksessa valojen käytöllä on myös suuri merkitys verteksin kokonaismäärään. Pikselivalot saattavat lisätä sitä, kuinka monesti monikulmioverkko piirretään. Monikulmioverkon piirtäminen useaan kertaan lisää piirtokomentoja, prosessoitavien verteksin lukumäärää sekä piirrettävien pikseleiden määrää (McDermott 2010, 32).



Leipomisella voidaan tarkoittaa 3D-grafiikan tuotannossa useita eri asioita. Leipominen on tapahtumasarja, jossa jokin dynaamisesti laskettava muutetaan staattiseksi. Staattiseksi muutettua informaatiota ei voida muuttaa, ellei leipomista suoriteta uudestaan. (Goldstone 2009, 146.)

Valaistuksen leipominen on prosessi, jossa otetaan talteen valaisun tuottamat väriarvot. Valaisun väriarvot saadaan valmiissa ympäristössä olevien objektien valaisusta ja varjostamisesta. Lopulta uudet väriarvot maalataan objektien pinnoille ja näin saadaan tuotettua ympäristöön leivottu valaistus. Valaistuksen leipominen on hyödyllistä, sillä sen käyttäminen on paljon nopeampaa kuin dynaamisesti laskettavien valojen. Kun valaisu leivotaan tarvitsee näytönohjaimen vain piirtää uusi tekstuuri objektien pinnalle (Goldstone 2009, 146).

#### 4.7 Sovelluksen kuvakkeet

Kun iOS-laitteille tehdään sovellus, täytyy sille myös luoda kuvake. Sovelluksen kuvake on kuva, joka tulee iOS-laitteen koti-valikkoon sovelluksen lataamisen jälkeen. Jokaiselle sovellukselle täytyy tehdä kuvake (Apple Inc 2012c). Kuvakkeen pitää täyttää tietyt kriteerit, jotta iOS voi näyttää sen oikein. Kuvakkeen koko riippuu sovelluksen kohdelaitteesta. Jos sovellus toimii usealla eri iOS-laitteella, täytyy kuvakkeesta tehdä erikokoisia versioita (taulukko 4). Sovelluksen ollessa peli, joka käyttää Applen Game Center -palvelua, käytetään kuvaketta myös siellä (Apple Inc 2012c).

TAULUKKO 4. Sovelluksen käynnistyskuvake koot (Apple Inc 2012c).

Laite	Pikseli koko
iPhone	57 x 57
iPhone korkea resoluutio	114 x 114
iPad	72 x 72
iPad korkea resoluutio	144 x 144

Applon App Store -kauppapaikkaan täytyy tehdä kuvakkeesta suuremmat versiot (taulukko 5). Kuvake, joka toimitetaan App Store -kauppapaikkaan, voi sisältää enemmän

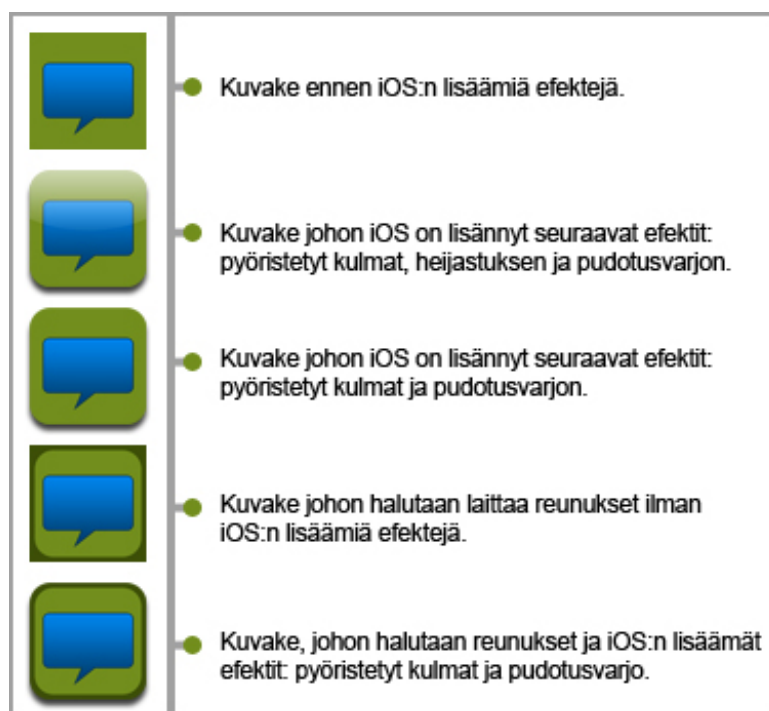
yksityiskohtia kuin laitteella käytettävä kuvake. On kuitenkin tärkeää, että kuvakkeet muistuttavat toisiaan lähes täydellisesti. (Apple Inc 2012c.)

TAULUKKO 5. Sovelluksen kuvake koot App Store -kauppapaikkaan (Apple Inc 2012c).

Laite	Pikseli koko
iPhone	512 x 512
iPhone korkea resoluutio	1024 x 1024
iPad	512 x 512
iPad korkea resoluutio	1024 x 1024

Kun iOS näyttää kuvakkeen laitteen näytöllä, lisää se siihen automaattisesti seuraavat efektit: pyöristetyt kulmat, pudotusvarjostuksen ja heijastavan kiillon. Heijastava kiilto on ainoa efekti, joka voidaan haluttaessa estää. Oikeanlaisessa kuvakkeessa täytyy olla 90 asteen kulmat, eikä se saa sisältää pudotusvarjoa, läpinäkyvyyttä tai heijastusta, ellei iOS:n automaattisesti asettamaa heijastusta estetä. (Apple Inc 2012c.)

Kuvan 9 tarkoituksena on esittää, miten iOS-laitteet näyttävät kuvan käynnistyskuvakkeena.



KUVA 9. Käynnistyskuvakkeen käyttäytyminen iOS-laitteilla (Mukaiillen Apple Inc 2012d, 11–12)

## 5 OPINNÄYTETYÖN TOTEUTUS

### 5.1 Opinnäytetyön tarkoitus ja prosessi

Opinnäytetyön tarkoituksena oli luoda opas mobiilipeligrafiikan tuottamisesta Ifelse Media Oy:lle. Oppaan tarkoitus on kuvata yrityksen standardit mobiiligrafiikan tuottamisessa yleisellä tasolla sekä huomioiden erityisesti Unity-ohjelman ja iOS-laitteiden tuomat rajoitukset sekä mahdollisuudet. Tarkoituksena on siis luoda selkeä dokumentaatio siitä, miten grafiikka pitäisi teknisesti toteuttaa.

Ifelse Media Oy perustettiin syyskuussa 2011. Vuodenlopulla yritykselle haluttiin luoda standardit mobiilipeligrafiikan tuottamiseen, jotta grafiikka peliprojekteissa olisi yhdenmukaista. Oppaan toteutus alkoi muistiinpanoja tekemällä ja aihealueen teorian tiedon keräämisellä. Merkittävimmässä asemassa olivat luonnollisesti Ifelse Media Oy:n kanssa käydyistä neuvotteluista nousseet tarpeet. Oppaan aineisto muodostuu siis teoriapohjasta sekä Ifelse Media Oy:n määrittelemistä lähtökohdista. Aineiston kerääminen aloitettiin alkuvuodesta 2012 ja opas kirjoitettiin kevään aikana.

### 5.2 Opas

Hyvin toteutetussa oppaassa on mietitty kohderyhmää, ja millaista mielikuvaa tahdotaan teoksella viestiä. Oppaan pitää olla yksilöllisen ja persoonallisen näköinen, jotta se erottuu edukseen muista vastaavista teoksista. Oppaassa merkittävimpinä kriteereinä voidaan pitää teoksen uutta muotoa, käytettävyyttä kohderyhmässä, sisällön sopivuutta kohderyhmän tarpeisiin, tuotteen kiehtovuutta, informatiivisuutta sekä selkeyttä ja johdonmukaisuutta. Lähdekritiikki on oppaan tekemisessä myös todella tärkeää. Oppaan toteutusmuotoa on syytä miettiä, jotta teoksesta olisi mahdollisimman paljon hyötyä kohderyhmälle. (Vilka & Airaksinen 2003, 51–53.)

Oppaan kohderyhmä oli heti työn alusta alkaen tiedossani, joten sen suurempia selvityksiä kohderyhmästä ei tarvinnut tehdä. Opas päätettiin toteuttaa PDF-muodossa, jotta se olisi mahdollisimman helposti kohderyhmän saatavilla. Tiedostoa tullaan jakamaan yrityksen Internet-sivuilla. Opasta ei aiota painaa kustannussyistä ja sen vuoksi, että

opasta tullaan päivittämään aina, kun Unity-ohjelmaan tulee merkittäviä päivityksiä tai silloin, kun Apple julkaisee iOS-laitteen, joka vaikuttaa oppaan sisältöön.

Opasta tehdessä on syytä puntaroida sen kokoa ja typografiaa, sillä oppaan koko ja siihen valittu typografia vaikuttavat sen luettavuuteen. Tekstin koolla on myös suuri merkitys oppaan luettavuuteen. Jos opas painetaan, täytyy sen laajuutta, kuvia ja värien käyttöä miettiä, sillä ne vaikuttavat oleellisesti oppaan valmistuskustannuksiin. Jos oppaan toimeksiantajana on yritys, tulee teoksessa huomioida myös yrityksen ulkoasuvaatimukset. Yritys saattaa haluta esimerkiksi, että teoksessa käytetään heidän typografisia ohjeita, värimaailmaa tai logoa. (Vilkka & Airaksinen 2003, 52–53.)

Oppaan kooksi valittiin A4, jotta sen tulostaminen olisi tarvittaessa mahdollista ja mahdollisimman vaivatonta. Typografia valinnoilla pyrittiin saavuttamaan mahdollisimman hyvä luettavuus käytettiin opasta sitten joko tietokoneella tai tulostettuna. Teoksessa käytettiin yrityksen logoa ja värimaailmaa.

Oppaan tekstin olisi hyvä puhutella sen kohderyhmää ja siinä tulisi käyttää sisällön kannalta asiaankuuluvaa kirjoitustyyliä. Tekstiä kirjoittaessa on syytä pohtia, mihin tarkoitukseen opas tehdään sekä kohderyhmän aiempaa tietämystä teoksen sisältämästä aiheesta. (Vilkka & Airaksinen 2003, 129.)

## 6 POHDINTA

Opinnäytetyöni lopputuloksena syntyi opas mobiilipeligrafiikan toteuttamiseen, jossa huomioidaan Unity-ohjelman ja iOS-laitteiden vaatimukset. Oppaassa esiteltiin teknii-koita, joiden avulla voidaan toteuttaa mobiilipeligrafiikkaa tämän hetkisille iOS-laitteille.

Opas onnistui mielestäni hyvin ja sen sisältö vastaa sille annettuja vaatimuksia. Työssä on käytetty vain luotettavia lähteitä. Lähteiden kanssa oli työn alussa myös haasteita, sillä sopivaa kirjallisuutta ei aluksi tahtonut löytyä. Välillä myös ajankäyttö osoittautui haasteelliseksi, sillä aloitin työn hieman myöhässä ja opintojen sekä työtehtävien yh-teensovittaminen ei aina ollut helppoa.

Oppaan toimivuutta päästään testaamaan tulevan kesän aikana, kun opas annetaan yri-tyksen harjoittelijan käyttöön. Opas aiotaan myös julkaista yrityksen Internet-sivuilla ja sitä tullaan kehittämään saadun palautteen perusteella. Opasta päivitetään myös aina silloin, kun Unity julkaisee ohjelmastaan sellaisen version, joka vaikuttaa oppaan sisäl-töön tai aina silloin, kun Apple julkaisee sellaisen laitteen, jolla on vaikutusta oppaa-seen. Saavutin myös henkilökohtaiset oppimistavoitteeni ja tällä hetkellä osaan tehdä teknisesti huomattavasti parempaa mobiilipeligrafiikkaa.

Vaikka olenkin saanut oppaan vasta valmiiksi, on minulle syntynyt jo uusia ideoita sen kehittämiseksi. Yrityksemme on nimittäin suunnitellut pelien julkaisemista Android-laitteilla ja opasta olisi hyvä täydentää vastaamaan näiden laitteiden vaatimuksia. Opas-ta voisi jatkossa täydentää myös esimerkiksi kertomalla 2D-grafiikan animaatiotekni-koista. Nähtäväksi jää, kuinka opas tulee elämään muutosten tuulessa.

## LÄHTEET

- Apple Inc. 2010. iPad Technical Specifications. Julkaistu: 4.3.2010. Luettu: 23.5.2012. <http://support.apple.com/kb/SP580>
- Apple Inc. 2012a. Compare iPad Models. Julkaistu 2012. Luettu: 26.4.2012. <http://www.apple.com/ipad/compare/>
- Apple Inc. 2012b. Compare iPhone Models. Julkaistu 2012. Luettu: 25.4.2012. <http://www.apple.com/iphone/compare-iphones/>
- Apple Inc. 2012c. Custom Icon and Image Creation Guidelines. Päivitetty 7.3.2012. Luettu: 16.3.2012. <http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobil ehig/IconsImages/IconsImages.html>
- Apple Inc. 2012d. iTunes Connect Developer Guide 7.4. Julkaistu: 29.3.2012. Luettu: 5.4.2012. [https://itunesconnect.apple.com/docs/iTunesConnect\\_DeveloperGuide.pdf](https://itunesconnect.apple.com/docs/iTunesConnect_DeveloperGuide.pdf)
- Corbo, Ruben. 2011. A Brief History of Mobile Gaming. Julkaistu 21.8.2011. Luettu: 24.4.2012. <http://www.seekomega.com/2011/08/a-brief-history-of-mobile-gaming/>
- Dilger, Daniel Eran. 2011. Inside Apple's iPad 2 A5: fast LPDDR2 RAM, costs 66% more than Tegra 2. Julkaistu: 13.3.2011. Luettu: 26.5.2012. [http://www.appleinsider.com/articles/11/03/13/inside\\_apples\\_ipad\\_2\\_a5\\_fast\\_lpddr2\\_ram\\_costs\\_66\\_more\\_than\\_tegra\\_2.html](http://www.appleinsider.com/articles/11/03/13/inside_apples_ipad_2_a5_fast_lpddr2_ram_costs_66_more_than_tegra_2.html)
- Duggan, Michael. 2008. 2D game building for teens. Clifton Park, N.Y.: Delmar; London: Cengage Learning.
- Gahan, Andrew. 2009. Game art complete: all-in-one: learn maya, 3ds max, zbrush, and photoshop winning techniques. Oxford: Focal.
- Goldstone, Will. 2009. Unity Game Development Essentials. Birmingham: Packt Publishing Ltd.
- Kim, Arnold. 2010. iPhone 4 Confirmed to Have 512MB of RAM. Julkaistu: 17.6.2010. Luettu: 26.5.2012. <http://www.macrumors.com/2010/06/17/iphone-4-confirmed-to-have-512mb-of-ram-twice-the-ipad-and-3gs/>
- Lehtovirta Pekka & Nuutinen Kari. 2000. 3D-sisältötuotannon peruskirja. Jyväskylä: Docendo Finland Oy.
- McDermott, Wes. 2010. Creating 3D game art for the iPhone with unity : featuring modo and Blender pipelines. Oxford: Focal.
- Pirkanmaan ELY-keskus ja Tekes. 2011. Peliteollisuus porskuttaa Pirkanmaalla. Julkaistu 28.10.2011. Luettu: 2.5.2012. [http://www.ely-keskus.fi/fi/tiedotepalvelu/2011/Sivut/Peliteollisuusporskuttaa\\_Pirkanmaalla.aspx](http://www.ely-keskus.fi/fi/tiedotepalvelu/2011/Sivut/Peliteollisuusporskuttaa_Pirkanmaalla.aspx)
- Puhakka, Antti. 2008. 3D-grafiikka. Helsinki: Talentum.

Slivka, Eric. 2012. Benchmarks on Third-Generation iPad Reveal 1 GHz CPU, 1 GB RAM. Julkaistu: 13.3.2012. Luettu: 25.5.2012.  
<http://www.macrumors.com/2012/03/13/benchmarks-on-third-generation-ipad-reveal-1-ghz-cpu-1-gb-ram/>

Steinbock, Dan. 2005. The mobile revolution: the making of mobile services world-wide. London; Sterling, VA: Kogan Page, cop.

TechTerms.com. 2009. Graphics. Päivitetty: 1.4.2009. Luettu: 26.5.2012.  
<http://www.techterms.com/definition/graphics>

Unity Technologies. 2010a. Materials and Shaders. Päivitetty: 16.9.2010. Luettu: 2.4.2012. <http://unity3d.com/support/documentation/Manual/Materials.html>

Unity Technologies. 2010b. How do I import objects from my 3D app? Päivitetty: 13.7.2010. Luettu: 28.3.2012.  
<http://unity3d.com/support/documentation/Manual/HOWTO-importObject.html>

Unity Technologies. 2011. Optimizing Graphics Performance. Päivitetty: 3.11.2011. Luettu: 17.4.2012.  
<http://unity3d.com/support/documentation/Manual/iPhone%20Optimizing%20Graphics%20Performance.html>

Unity Technologies. 2012a. Draw Call Batching. Päivitetty: 2.2.2012. Luettu: 6.4.2012.  
<http://unity3d.com/support/documentation/Manual/iphone-DrawCall-Batching.html>

Unity Technologies. 2012b. Meshes. Päivitetty: 19.1.2012. Luettu: 16.4.2012.  
<http://unity3d.com/support/documentation/Components/class-FBXImporter.html>

Vilkka Hanna & Airaksinen Tiina. 2003. Toiminnallinen opinnäytetyö. Helsinki: Tammi.

Watkins, Adam. 2011. Creating Games with Unity and Maya: How to Develop Fun and Marketable 3D Games. Oxford: Focal.

White, Joe. 2011. According To chAIR Entertainment, Apple's iPhone 4S Has 512 Megabytes Of RAM. Julkaistu: 6.10.2011. Luettu: 25.5.2012.  
<http://appadvice.com/appnn/2011/10/apples-iphone-4s-has-512-megabytes-of-ram-just-like-iphone-4-and-ipad-2>

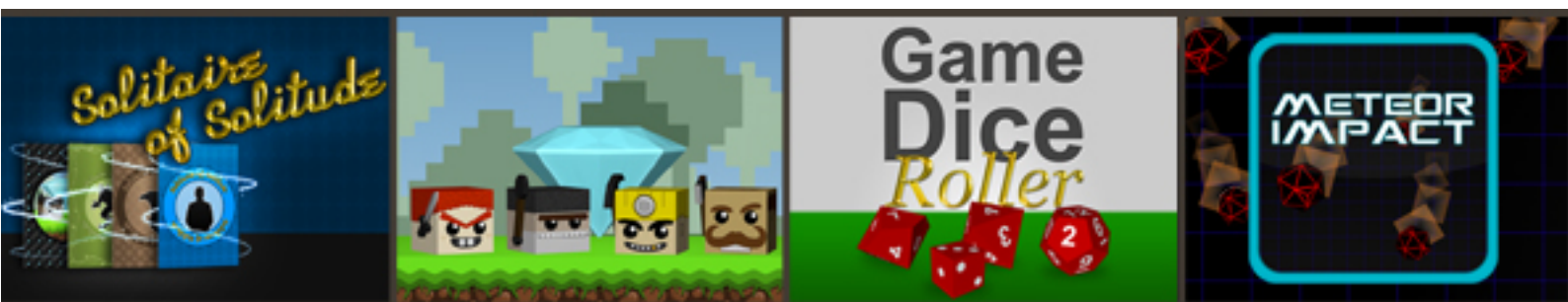
**LIITTEET**

## Liite 1. Mobiilipeligrafiikkaopas

1 (25)

**MOBIILIPELIGRAFIIKKAOPAS**

3.5.2012





## Sisällys

1 GRAFIikka UNITY-OHJELMASSA JA IOS-LAITTEILLA .....	3
2 YLEISTÄ IOS-LAITTEISTA .....	4
3 YLEISTÄ UNITY-OHJELMASTA .....	6
3.1 PIIRTOKOMENNOT .....	6
3.1.1 PIIRTOKOMENTOJEN DYNAAMINEN YHDISTÄMINEN .....	6
3.1.2 PIIRTOKOMENTOJEN STAATTINEN YHDISTÄMINEN .....	8
3.2 3D-MALLIT .....	8
3.3 3D-PELIHAHMOT .....	10
3.4 LUUSTO JA NIVELISTÖ .....	10
3.5 3D-HAHMOANIMAATIO .....	11
3.6 SKINNED MESH RENDERER -KOMPONENTTI .....	12
3.7 MATERIAALIT JA SHADERIT .....	13
3.8 TEKSTUURIT .....	14
3.9 KUUTIOKARTTA-TEKSTUURIT .....	17
3.10 SKYBOX TEKSTUURIT .....	17
3.11 NORMAALIKARTAT .....	17
3.12 KORKEUSKARTAT .....	18
3.13 TEKSTUURIATLAKSET .....	18
3.14 TEKSTUURIEN PAKKAAMINEN UNITY-OHJELMASSA .....	19
3.15 VALAISU .....	19
4 KANSIORAKENNE JA NIMEÄMISKÄYTÄNTÖ .....	21
5 SOVELLUSKUVAKE .....	22
5.1 GAME CENTER .....	23
6 FONTIT .....	24
7 3D-MALLIEN TYÖVÄLINEET JA TALLENNUSMUODOT .....	24
8 MUISTILISTA MOBIILIPELIGRAFIIKAN OPTIMOINTIIN .....	25

## **1 GRAFIIKKA UNITY-OHJELMASSA JA IOS-LAITTEILLA**

Peligrafiikan tekemisessä täytyy ottaa huomioon kohdelaitteet sekä ohjelmat, joilla pelit toteutetaan. Oppaan tarkoituksena on antaa grafiikan tekijälle kattava kuvaus siitä millaista grafiikkaa yritykselle pitää teknisesti toteuttaa. Mahdollistaaksemme sen, että peli toimii iOS-laitteilla hyvin, täytyy kohdelaitteiden vaatimukset ottaa huomioon grafiikan tuotantovaiheessa.

Pelien tekeminen iOS-laitteille vaatii erilaista lähestymistapaa kuin mitä käytettäisiin PC-pelien tekemiseen. Toisin kuin PC-laitteet, iOS-laitteet ovat standardoitu, eivätkä ne ole niin tehokkaita tai nopeita kuin PC-laitteet. Tämän vuoksi pelienkehitystä pitää lähestyä iOS-laitteilla hieman eri tavalla. Myös Unity-ohjelman ominaisuudet ovat iOS-laitteilla hieman erilaiset kuin PC-laitteille tehdessä.

Ifelse Media Oy on määrittänyt, että yrityksen tekemien sovellusten ja pelien on toimittava vähintään iPhone 3gs -laitteella. Tämän takia useat oppaassa käsiteltävät asiat on päätetty käsitellä iPhone 3gs -laitteen suorituskyvyn puitteissa. Tässä oppaassa termillä iOS-laitteet tarkoitetaan seuraavia Applen tuotteita: iPhone 3gs, iPhone 4, iPhone 4S, iPad, iPad 2 sekä kolmannen sukupolven iPad.

## 2 YLEISTÄ IOS-LAITTEISTA

iOS-laitteiden grafiikkaprosessorit käyttävät Tiled Based Deferred Rendering - grafiikkarendausta. Toisin kuin useimmat pöytäkoneiden grafiikkaprosessorit, keskittyy iOS-laitteiden grafiikkaprosessori minimoimaan kuvan näkymään renderöintiin tarvittavan ajan. Tällä tavalla ainoastaan näkyvät pikselit kuluttavat renderöinnin käsittelyresursseja.

Grafiikkaprosessorin ruutupuskuri on jaettu tiiliin ja renderöinti tapahtuu tiili kerrallaan. Ensimmäisenä kaikki ruudun kolmiopolygonit kerätään ja määritetään tiiliin. Tämän jälkeen jokaisen näkyvän kolmiopolygonin sirpaleet valitaan. Lopuksi valitut kolmiopolygonin sirpaleet viedään rasterointiin. Tässä vaiheessa hylätään polygonin sirpaleet, jotka ovat piilossa kameran näkymästä.

Toisin sanoen iOS-laitteiden grafiikkaprosessori käyttää toteutuksessa piilopintojen poisto -operaatiota pienentääkseen renderöimiseen tarvittavia kustannuksia. Tällainen arkkitehtuuri kuluttaa vähemmän muistin kaistanleveyttä, omaa pienemmän virrankulutuksen sekä hyödyntää tekstuurien välimuistia paremmin. TBDR-grafiikkarendauksen avulla laite pystyy hylkäämään piilossa olevia fragmentteja ennen varsinaista rasterointia. Tämän tekniikan avulla pystytään pitämään myös päällepiirtäminen (*overdraw*) alhaisena.

Grafiikkaa tehdessä on hyvä tuntea hieman kohdelaitteistoa. Näyttöjen resoluutiot ovat yksi tärkeimmistä laitteen ominaisuuksista grafiikan tekemisen kannalta. Laitteiden resoluutiot on tärkeää tuntea esimerkiksi tekstuureja tehdessä. Seuraava taulukko esittelee iOS-laitteiden resoluutiot.

Laitteen nimi	Resoluutio
iPhone 3GS	480x320, 163 ppi
iPhone 4	960x640, 326 ppi
iPhone 4S	960x640, 326 ppi
iPad	1024x768, 132 ppi
iPad 2	1024x768, 132 ppi
iPad kolmas sukupolvi	2048x1536, 264ppi

5 (25)

Kuvan 1 avulla pyritään selventämään, kuinka paljon laitteiden resoluutioilla on eroa.



Kuva 1. iOS-laitteiden resoluutiot. Kuva skaalattu neljä kertaa todellista kuvaa pienemmäksi.

Näytön suuri resoluutio voi aiheuttaa ongelmia, sillä iOS-laitteista iPhone 3gs, iPhone 4 ja iPad käyttävät samaa grafiikkaprosessoria. Grafiikkaprosessorin täyttönopeutta voidaan pitää iPhone 4 - ja iPad-laitteilla rajoitettuna.

Täyttönopeudella tarkoitetaan sitä, kuinka nopeasti laitteen grafiikkaprosessori kykenee renderöimään pikseleitä sekunnissa laitteen näytölle. Käytännössä tämä tarkoittaa sitä, että iPhone 4 - ja iPad-laitteilla grafiikkaprosessori joutuu tekemään noin 4–5 kertaa enemmän töitä grafiikan piirtämiseksi. Suurimmat ongelmat saattavat näillä laitteilla tulla esille läpinäkyvien pintojen piirtämisessä, sillä laitteet joutuvat piirtämään ne useampaan kertaan.

### 3 YLEISTÄ UNITY-OHJELMASTA

Jokaisen 3D-pelin ydin on monikulmioverkoissa (*mesh*) eli objekteissa, jotka koostuvat polygoneista, joissa sovelletaan tekstuureja. Unity-ohjelmassa ei ole omaa työkalua monikulmioverkkojen tekemiseen, mutta se toimii hyvin useiden tunnettujen 3D-mallinnusohjelmien kanssa. Monikulmioverkkojen tuonti 3D-mallinnusohjelmasta Unity-ohjelmaan on melko yksinkertaista.

Unity-ohjelmassa monikulmioverkot renderöidään laitteen näytölle renderöinti-komponenteilla. Renderöinti-komponentteja on ohjelmassa useita, mutta Mesh Renderer -komponentti on niistä useimmiten käytetty. Monikulmioverkkojen ulkoasua kontrolloidaan renderöinti-komponentin materiaalin avulla.

#### 3.1 PIIRTOKOMENNOT

Unity-ohjelmassa piirretään objekti laitteelle piirtokomentojen (*draw calls*) avulla. Jokainen erillinen objekti vaatii oman piirtokomennon ellei piirtokomentoja yhdistetä. Jos piirtokomentoja tehdään liian paljon, saattavat ne vaikuttaa kohdelaitteen prosessorin suorituskykyyn. Käytännössä tämä tarkoittaa sitä, että pelin kehysnopeus tippuu laitteella liian pieneksi ja pelattavuus kärsii oleellisesti. Tämän takia piirtokomentoja on syytä yhdistää niin paljon kuin mahdollista.

Piirtokomentoja voidaan Unity-ohjelmassa yhdistää kahdella eri tavalla, dynaamisella ja staattisella. Staattiseen yhdistämiseen tarvitaan Unity Pro - ja iOS Pro -ohjelmalisenssit, joita ei vielä tällä hetkellä ole yrityksellä käytössä.

##### 3.1.1 PIIRTOKOMENTOJEN DYNAAMINEN YHDISTÄMINEN

Objektien täyttäessä tietyt kriteerit yhdistää Unity-ohjelma piirtokomentoja dynaamisesti. Dynaaminen yhdistäminen tapahtuu automaattisesti eikä se vaadi käyttäjältä ylimääräistä työtä. Objektien pitää jakaa sama materiaali, jotta Unity-ohjelma voi yhdistää objekteja samaan piirtokomentoon dynaamisesti. Saavuttaaksemme taas mahdollisimman tehokkaan yhdistämisen, tulee mahdollisimman monen objektin jakaa sama materiaali. Jos käytössä on materiaaleja, jotka eroavat toisistaan vain tekstuuriltaan, voidaan tekstuurit

yhdistää yhdeksi isoksi tekstuuriksi. Tällaista tekstuuria kutsutaan tekstuuriatlakseksi. Kun tekstuurit ovat samassa atlaksessa, voidaan objekteissa käyttää vain yhtä materiaalia.

Objektien verteksien määrällä on myös vaikutusta siihen, voidaanko piirtokomentoja yhdistää dynaamisesti. Dynaaminen yhdistäminen on mahdollista objekteille, jotka sisältävät alle 900 verteksiä. Verteksien maksimimäärän voi laskea yksinkertaisella laskutoimitukselle, joka menee seuraavasti:  $900 / \text{verteksien attribuuttien määrä}$ . Esimerkiksi, jos vertekseillä on kolme attribuuttia, on mallin verteksien maksimimäärä 300 verteksiä. Tyypillisesti dynaaminen yhdistäminen on mahdollista objekteille, jotka sisältävät alle 225 verteksiä. Verteksien maksimimäärään vaikuttaa myös se, millaista shaderia objekti käyttää. Shaderin ominaisuudet vaikuttavat seuraavasti verteksien maksimimäärään. Jos shaderi käyttää verteksien sijaintia, normaaleja ja yhtä UV-karttaa, voidaan yhdistää objektit, jotka sisältävät maksimissaan 300 verteksiä. Jos taas shaderi käyttää verteksien sijaintia, normaaleja, kahta UV-karttaa ja tangenttia, voidaan yhdistää objektit, jotka sisältävät korkeintaan 180 verteksiä. Alla olevassa taulukossa on raja-arvot tyypillisimmistä tapauksista.

Huomioitavaa	Attribuutit	Verteksien määrä
Pelkkä väri, ei tekstuuria	pos, normal	450
Ei valokarttaa	pos, normal, UV	300
Unity-ohjelman oletus shaderi	pos, normal, UV, UV2	225
Bumpmap, ei valokarttaa	pos, normal, UV, tangent	225
Bumpmap ja valokartta	pos, normal, UV, UV2, tangent	180

Objektit, jotka käyttävät valokarttoja, omaavat piilotetun materiaaliparametrin. Näin ollen valokarttaa käyttäviä objekteja ei voida yhdistää, elleivät ne viittaa valokartassa samaan osaan. Myös multi-pass shaderien käyttäminen rikkoo dynaamisen piirtokomentojen yhdistämisen.

### 3.1.2 PIIRTOKOMENTOJEN STAATTINEN YHDISTÄMINEN

Staattisessa yhdistämisessä objektit eivät saa liikkua ja niiden pitää käyttää samaa materiaalia. Se sopii hyvin tausta objekteille, jotka eivät pelissä liiku, käänny tai muuta kokoaan. Objekteilla ei ole staattisessa yhdistämisessä verteksirajoituksia.

Staattinen yhdistäminen vaikuttaa huomattavasti tehokkaammin kuin dynaaminen yhdistäminen, sillä se vaatii prosessorilta vähemmän tehoa. Se vaatii kuitenkin enemmän muistin käyttöä yhdistettyjen geometrioiden tallentamiseen. Jos useat objektit jakavat saman geometrian ennen staattista yhdistämistä, jokaisen objektin geometriasta luodaan kopio. Tämä ei välttämättä ole hyvä idea, sillä joskus on parempi uhrata renderöintiin tarvittavaa suorituskykyä välttääksemme muistin liiallista käyttöä. Esimerkiksi metsän muodostaminen samanlaisilla puilla, jotka ovat yhdistetty staattisesti, saattaa vaikuttaa muistiin liikaa. Staattinen yhdistäminen on mahdollista vain Unity iOS Pro -ohjelma versiossa.

Staattista yhdistämistä voidaan tehdä myös 3D-mallinnusohjelmassa. Tämä tarkoittaa käytännössä sitä, että yhdistetään objekteja, jotka eivät pelin aikana liiku, ja joiden tiedetään esiintyvän yhtä aikaa ruudulla. Tällä tavalla yhdistetyillä objekteilla täytyy olla yhteinen tekstuuri. Objektien yhdistämistä voidaan myös tehdä jälkikäteen Unity-ohjelmassa eräällä editoriskriptillä.

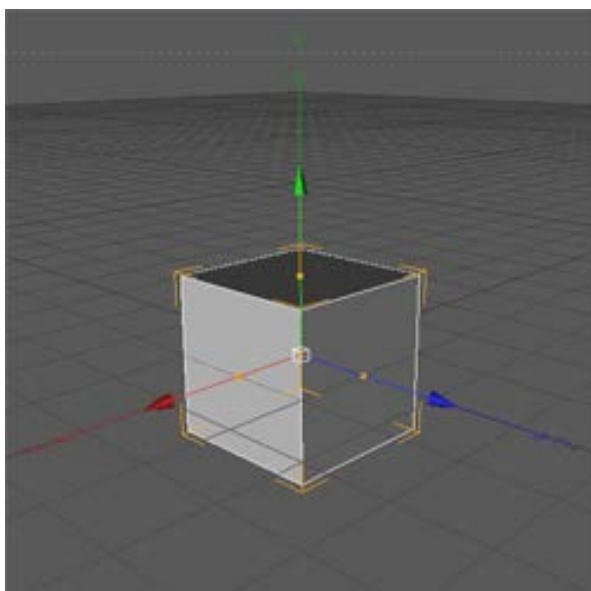
### 3.2 3D-MALLIT

Tehtäessä peligrafiikkaa, 3D-mallit eivät saa sisältää liikaa polygoneja. Toisin sanoen, kun 3D-malleja tehdään, on syytä käyttää polygoneja niin vähän kuin on tarpeellista. Unity-ohjelma muuttaa automaattisesti nelisivuiset polygonit kolmisivuisiksi polygoneiksi. Yli nelisivuiset polygonit saattavat aiheuttaa ongelmia Unity-ohjelmassa. Esimerkiksi yli nelisivuiset polygonit saattavat aiheuttaa sen, että kun Unity-ohjelma muuttaa ne kolmisivuisiksi, osa polygonien pinnoista on väärinpäin tai polygonit eivät yhdisty toisiinsa halutunlaisesti. Tämän vuoksi on syytä muuttaa polygonit kolmisivuisiksi jo 3D-mallinnusohjelmassa.

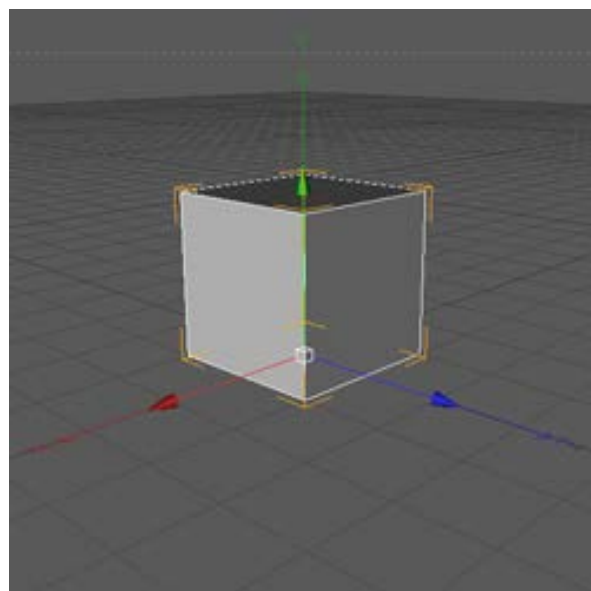
Yksittäisellä kuvaruudulla objektien yhteenlaskettu verteksien määrä ei saa ylittää 40 000 verteksiä. Kannattaa huomioida, että todellinen verteksien määrä, jonka

grafiikkalaitteisto käsittelee, ei ole yleensä sama määrä, minkä mallinnusohjelma ilmoittaa. Mallinnusohjelmat yleensä näyttävät geometrian verteksien määrän eli kulmapisteiden määrän, jotka muodostavat mallin. Näytönohjain joutuu jakamaan jotkut geometriset verteksit kahdeksi loogisemmaksi verteksiksi renderöintiä varten. Verteksi pitää jakaa, jos sillä on useampi normaali, UV-koordinaatti tai verteksiväri. Niinpä verteksien määrä Unity-ohjelmassa saattaa olla paljon suurempi kuin mitä 3D-mallinnusohjelma ilmoittaa.

Objektit ovat syytä nimetä huolella 3D-mallinnusohjelmassa, sillä se helpottaa niiden käsittelyä Unity-ohjelmassa. Objektit pitää myös keskittää 3D-mallinnusohjelmassa objektikoordinaatistoon. Keskittämisen ideana on se, että objekteja olisi helpompi käsitellä Unity-ohjelmassa.



*Kuva 3. Objekti 3D-mallinnusohjelmassa keskitettynä objektikoordinaatistoon.*



*Kuva 2. Objekti 3D-mallinnusohjelmassa keskitettynä objektikoordinaatistoon ja y-akseli laitettu objektikoordinaatiston nollakohtaan*

Joskus on suositeltavaa, että objekti laitetaan Y-akselilla objektikoordinaatistossa nollakohtaan. Esimerkiksi pöydän asettaminen lattialle Unity-ohjelmassa on helpompaa, kun objekti ollaan keskitetty ja se on asetettu Y-akselilla objektikoordinaatistossa nollakohtaan.



### 3.3 3D-PELIHAHMOT

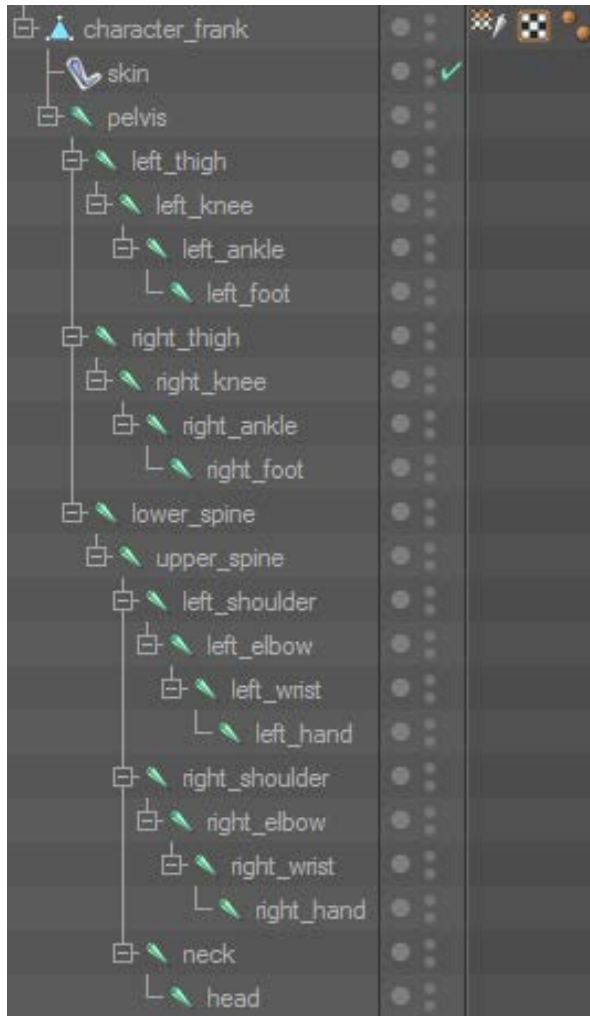
Peleissä käytettävien hahmojen verteksien maksimimäärä on 1500. Myös hahmojen mallintamisessa on hyvä muistaa, ettei ylimääräisiä polygoneja kannata käyttää turhaan. Hahmon polygonien määrä riippuu täysin pelistä eli, jos pelissä on useita hahmoja kerralla laitteen näytöllä, on syytä pitää polygonimäärä mahdollisimman alhaisena. Jos pelissä taas on vain muutama hahmo laitteen näytöllä yhtäaikaaisesti, voidaan polygonien määrää nostaa. Animaation tai luuston sisältävien monikulmioverkkojen piirtokomentoja ei voida yhdistää dynaamisesti eikä staattisesti.

### 3.4 LUUSTO JA NIVELISTÖ

Hahmossa ei saa olla luita eikä niveliä yli 30 kappaletta. Luita ja niveliä pitää käyttää niin vähän kuin on mahdollista, sillä mitä vähemmän luita tai niveliä käytetään, sitä parempi on laitteen suorituskyky. Luissa ja nivelissä on hyvä muistaa, että ne pitää nimetä riittävän kuvaavasti. Luut ja nivelet pitää nimetä englanniksi. Esimerkki luuston nimeämisestä: `left_elbow`. Jos esimerkiksi selkäranka koostuu useammasta nivelestä kuin esimerkikuvassa, tulee nivelet nimetä seuraavasti: `spine_1`, `spine_2`, `spine_3` jne.

Yrityksen käytössä olevassa Cinema 4D Prime R12 -ohjelmassa ei ole enää luu-työkalua. Luu-työkalu on kyseisessä versiossa korvattu nivel-työkalulla. Cinema 4D R11 -ohjelmaa vanhemmissa versioissa on suositellaan konvertoida nivelet luiksi ennen kuin 3D-hahmo viedään .fbx-muotoon.

Kuvassa neljä on esiteltynä yrityksen käytössä oleva perusnivistö ja nivelten nimeämiskäytäntö. Jos niveliä on hahmossasi enemmän, käytä kuvassa näkyvää nimeämistyyliä. Tärkeintä nivelten nimeämisessä on se, että nivelen nimi kuvaa mahdollisimman hyvin niveltä.



Kuva 4. Luiden nimeämiskäytäntö

### 3.5 3D-HAHMOANIMAATIO

Unity-ohjelma ei tue vielä tällä hetkellä Cinema 4D -ohjelman Point Level Animation -animaatiotekniikkaa. Tämän vuoksi kaikki animaatiot Cinema 4D -ohjelmassa pitää toteuttaa luuston tai nivelistön avulla, jotta animaatiot toimisi Unity-ohjelmassa.

Cinema 4D -ohjelmalla on suositeltavaa, että animaatiot toteutetaan forward kinematics -tekniikalla, koska Unity-ohjelma ei tue Cinema 4D -ohjelman inverse kinematics -tekniikalla tehtyjä animaatioita. Jos animaatiot toteutetaan inverse kinematics -tekniikalla, täytyy animaatiot leipoa forward kinematics -tekniikkaan ennen kuin ne viedään fbx-formaattiin.

Unity-ohjelma tukee joidenkin 3D-mallinnusohjelmien IK-tekniikalla tehtyjä animaatioita. Kun IK-tekniikalla toteutetut hahmot tuodaan Unity-ohjelmaan, leipoo ohjelma automaattisesti IK-animaation FK-animaatioksi.

Jos käytät 3D-mallinnusohjelmassa jotain muuta kuin Cinema 4D -ohjelmaa, voit lukea lisää muiden ohjelmien tuetuista animaatiotekniikoista Unity-ohjelman [ohjekirjasta](#).

Animaation sisältävän hahmon mukaan on liitettävä erillinen tekstitiedosto, jossa kerrotaan animaatioiden nimet ja avainruutujen välit. Esimerkiksi idle 0–9, walk 10–40, run 41–55, jump 56–60. Animaatio tai animaatioasetti voidaan myös tallentaa omaan tiedostoon. Jos näin halutaan tehdä, täytyy käyttää @-nimeämiskäytäntöä esimerkiksi character.fbx, character@idle.fbx ja character@walk.fbx.

Pidä Forward - ja Inverse kinematics -tekniikat erossa toisistaan. Kun animaatiot tuodaan Unity-ohjelmaan, mallin IK-solmut leivotaan FK:ksi ja sen seurauksena Unity-ohjelma ei tarvitse IK-solmuja ollenkaan. Jos ne kuitenkin on jätetty malliin, niin ne aiheuttavat prosessorin kuormittumista, vaikka ne eivät vaikutakaan animaatioon. Voit poistaa tarpeettomat IK-solmut joko Unity-ohjelmassa tai 3D-mallinnusohjelmassa. Parasta olisi pitää IK ja IF hierarkiat erillään mallintamisen aikana, jotta IK-solmut ovat tarvittaessa helppo poistaa.

### 3.6 SKINNED MESH RENDERER -KOMPONENTTI

Skinned mesh renderer -komponentti lisätään automaattisesti tuonnin yhteydessä, jos monikulmioverkko on skinnattu. Skinned mesh -komponenttia käytetään hahmon renderaamiseen.

Kun hahmon animaatio on toteutettu luita tai niveliä käyttäen, vaikuttavat yksittäiset luut tai nivelet osaan monikulmioverkkoa. Suurin hyöty käyttää luilla tai nivelillä animoitua hahmoa on se, että Unity-ohjelmassa voidaan mahdollistaa luiden altistuminen fysiikalle. Tällä tavoin hahmosta voidaan tehdä räsynukke (*dragdoll*).

Unity-ohjelma voi skinnata jokaisen verteksin joko yhteen, kahteen tai neljään luuhun tai niveleen. Neljän luun painotus näyttää parhaimmalta, mutta se on kaikkein kallein operaatio. Kahden luun painotus on hyvä kompromissi ja sitä voidaan useimmiten käyttää peleissä.

Oletusarvoisesti skinnatut monikulmioverkot eivät päivity, jos ne eivät ole näkyvissä. Skinnaus päivittyy vasta sitten, kun monikulmioverkko tulee takaisin näytölle. Tämä on hyvin tärkeää suorituskyvyn optimoimiseksi, sillä se mahdollistaa sen, että useat hahmot, jotka eivät ole näkyvissä, voivat juoksennella ympäriinsä viemättä yhtään prosessointitehoa.

Yksittäisessä hahmossa tulee käyttää vain yhtä skinned mesh renderer -komponenttia, sillä Unity-ohjelma optimoi animaatiot käyttäen visibility cullin - ja bounding volume updates -prosesseja. Nämä optimoinnit ovat ainoastaan aktiivisia, jos käytetään yhtä animaatio komponenttia yhdessä skinned mesh renderer -komponenttissa. Mallin renderöintiäika voi karkeasti kaksinkertaistua, jos käytetään kahta skinnattua monikulmioverkkoa yhden sijasta.

### 3.7 MATERIAALIT JA SHADERIT

Unity-ohjelmassa materiaaleilla ja shadereilla on läheinen yhteys. Shaderit sisältävät koodin, joka määrittelee millaisia ominaisuuksia ja asetteja käytetään. Materiaalit puolestaan mahdollistavat ominaisuuksien muokkaamisen ja asettien asettamisen.

Jokaisen monikulmioverkon materiaolimäärä on pyrittävä pitämään mahdollisimman vähäisenä. Hahmon materiaaleiksi pitäisi riittää korkeintaan 2–3 eri materiaalia. Useimmiten yksi materiaali riittää hahmolle. Joskus materiaolimäärä saattaa hahmoissa nousta, sillä esimerkiksi hahmon silmissä saatetaan haluta käyttää erilaista shaderia.

Unity-ohjelmassa on yli 40 eri shaderia, joiden avulla voidaan muokata grafiikan ulkoasua. Shaderi määrittää kaavan, miten pelissä varjostusten tulisi näkyä. Shadereilla on useita ominaisuuksia, yleensä tekstuureja. Shaderit otetaan käyttöön materiaaleissa, jotka liitetään suoraan yksittäisille peliobjekteille.

Shaderi määrittelee tavan, jolla objekti rendataan, joitain tekstuuri ominaisuuksia sekä väri- ja numeroasetuksia. Objektin renderöintimenetelmä riippuu käyttäjän laitteen näytönohjaimesta.

Materiaalille valitaan shaderi, jonka jälkeen sille määritetään ominaisuuksia. Yleensä ominaisuudet ovat tekstuureja ja värejä. Ominaisuudet, joita shaderissa käytetään, saattavat vaihdella. Materiaali määrittelee mitä tekstuureja rendauksessa käytetään, mitä värejä rendauksessa käytetään sekä muut assetit kuten kuutiokartta tekstuurit, jonka shaderi saattaa vaatia rendaukseen.

Toisin kuin pöytäkoneilla, iOS-laitteilla aiheutuu suorituskyvyssä ylikuormitusta alfa-testauksessa (*alpha-testing*). Alfa-testaus shaderit tulisivin korvata alfa-sekoitus (*alpha-blend*) shadereilla, jos se on mahdollista. Jos alfa-testaus shadereiden käyttöä ei voida välttää, tulee pyrkiä siihen, että alfa-testattavien pikselien määrä on alhainen.

### 3.8 TEKSTUURIT

Tekstuurien avulla saadaan monikulmioverkot, partikkelit ja käyttöliittymät hyvännäköisiksi. Tekstuurit ovat kuva- tai videotiedostoja, jotka asetetaan objektin päälle tai ne kiedotaan objektin ympärille. Koska tekstuurit ovat hyvin tärkeitä, on niillä paljon ominaisuuksia Unity-ohjelmassa.

Unity-ohjelmassa tekstuurin tuontiasetukset voidaan asettaa sen mukaan miten tekstuuria aiotaan käyttää. Valinnan jälkeen ohjelma laittaa tekstuurille käyttökohteesta riippuen perusparametrit. Jos tekstuuria halutaan hallita täysin, kannattaa tekstuurin tyyppiä valita "advanced". Unity-ohjelmassa on olemassa seuraavat tekstuurityypit:

- Texture - Yleisin asetus tekstuureilla.
- Normal Map - Muuttaa värikanavat sopivaan muotoon reaaliaikaisessa normal map -tekniikassa.
- GUI - Valitaan, jos tekstuuria halutaan käyttää käyttöliittymissä.
- Reflection - Tunnetaan myös Cupemap-tekstuurina, käytetään luomaan heijastukset teksturiin.

- Cookie - Asettaa perus parametrit tekstuurille, jota käytetään valojen evästeissä.
- Advanced - Kun halutaan asettaa täsmälliset parametrit tekstuuriin, ja kun halutaan hallita tekstuuria täysin.

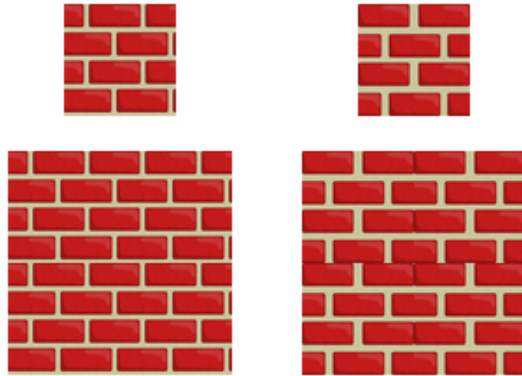
Tekstuurien resoluutioiden tulee olla tehtynä kahden potenssiin aina, kun se on mahdollista. Seuraavat tekstuuri koot ovat suositeltavia: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 ja 2048 pikseliä. Tekstuurien ei kuitenkaan tarvitse olla neliönmallisia, vaan ne voivat olla kooltaan esimerkiksi 1024x512.

Unity-ohjelmassa on kuitenkin mahdollista käyttää muitakin kuin kahden potenssissa olevia tekstuureja. Jos tekstuureja, joita ei ole tehty kahden potenssiin, käytetään mihinkään muuhun kuin käyttöliittymiin, ne muutetaan pakkaamattomaan RGBA 32 -bittiseen formaattiin. Jos tekstuureja ei ole tehty kahden potenssiin, tarvitaan niiden käsittelyyn enemmän muistia. Tämä tarkoittaa käytännössä sitä, että niiden lataaminen ja renderaaminen on hitaampaa. Tämän vuoksi on suositeltavaa, että tekstuurit tehtäisiin aina kahden potenssiin. Kuvien täytyy olla tehtyinä kahden potenssiin ja niiden on hyvä olla neliönmallisia, sillä PVRT-pakkausformaatti ei pakkaa muun mallisia tiedostoja.

Tekstuureja, joita käytetään muualla kuin käyttöliittymissä, voidaan pakata vaikka ne eivät olisikaan tehty kahden potenssiin. Niihin lisätään kuitenkin automaattisesti pikseleitä tuontiasetuksista riippuvalla tavalla. Eli esimerkiksi tekstuuri, joka on kooltaan 256x300 muuntuu tilanteesta riippuen, joko 256x256 tai 256x500 kokoiseksi. Tilanteissa, joissa halutaan saada esitettyä pikselintarkkaa grafiikkaa, kuten esimerkiksi käyttöliittymissä, saattaa automaattinen skaalaus tuottaa visuaalisia ongelmia. Joissain tapauksissa, esimerkiksi, jos tehdään tekstuuria pinnalle, jonka mittasuhteet on 1:3, voi olla perusteltua käyttää tekstuuria, jota ei olla tehty kahden potenssiin. Tuontiasetuksia säätämällä ongelmilta yleensä vältytään, mutta grafiikantekijän pitää muistaa erikseen ilmoittaa, että tekstuurin resoluutio ei ole kahden potenssiin.

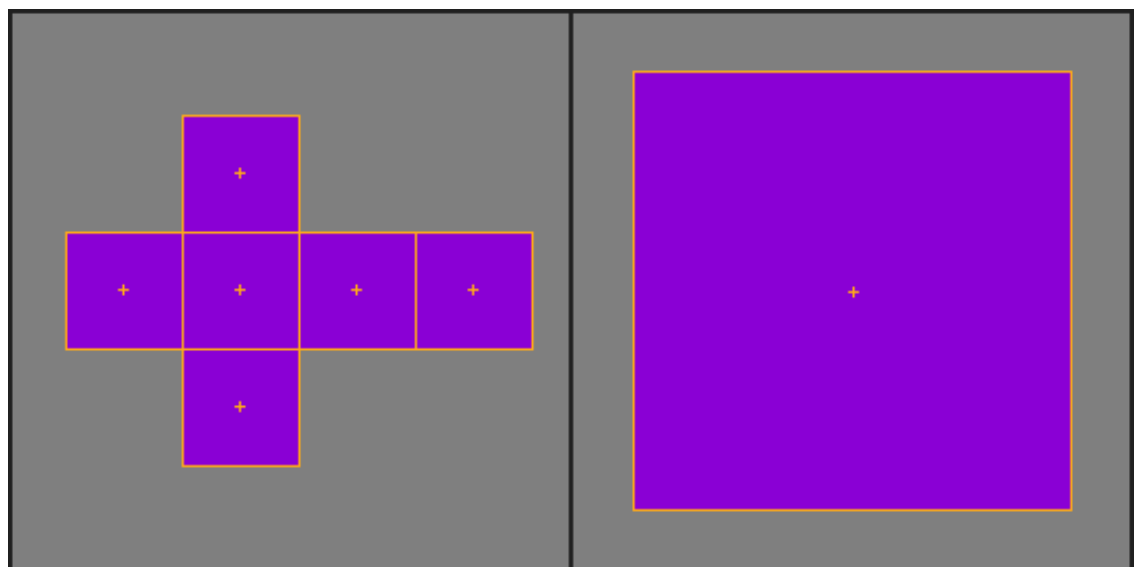
Myös toistuvien tekstuurien käyttämistä kannattaa harkita, sillä niiden avulla on mahdollista pienentää tekstuurien viemää tilaa projektissa. Oikeastaan toistuvia tekstuureja kannattaa käyttää melkein aina, jos siihen on mahdollisuus. Joissain

tilanteissa toistuvia tekstuureja on pakko käyttää. Esimerkiksi maastotekstuurien täytyy olla toistuvia tekstuureja.



*Kuva 5. Toistuvat tekstuurit. Kuvassa ylhäällä vasemmalla on tekstuuri, joka toistuu hyvin. Oikealla ylhäällä on tekstuuri, jonka toistamisessa voi huomata saumojen syntymisen.*

Cinema 4D -ohjelmassa käytetään mallien teksturointiin UV-kartoitustekniikkaa. Malleja teksturoidessa on pyrittävä pitämään UV-karttojen saumat ja kovat reunat mahdollisimman vähäisinä.



*Kuva 6. Kuution UV-kartat. Vasemmanpuoleisessa UV-kartassa syntyy saumat tekstuurin ulkoreunoille. Oikeanpuoleisessa UV-kartassa ei synny saumoja ollenkaan.*

### 3.9 KUUTIOKARTTA-TEKSTUURIT

Unity-ohjelmassa kuutiokartta-tekstuurit koostuvat kuudesta erillisestä neliötekstuurista. Tekstuurit laitetaan kuvitteellisen kuution pinnoille. Kuutiokartta-tekstuureja käytetään objekteissa yleensä luomaan heijastus kaukaisuudessa näkyvästä ympäristöstä samalla tavalla kuin skybox näyttää kaukaisen maiseman taustalla. Unity-ohjelman heijastus-shadereissa käytetään kuutiokartta-tekstuureja esittämään heijastuksia.

### 3.10 SKYBOX TEKSTUURIT

Skyboxit käyttävät kuutiokarttatekniikkaa pelin taustan luomiseen. Skybox rendataan laitteen näytölle ennen kuin mitään muuta rendataan. Sen tarkoituksena on antaa vaikutelma maiseman horisontista. Materiaali, jota käytetään skyboxien rendamiseen, sisältää myös kuusi tekstuuria. Tämän materiaalin pitäisi käyttää Skybox shaderia. Skyboxille on varattu kuusi tekstuuri paikkaa, jotka ovat jokaiseen pääsuuntaan (-/+X, -/+Y, -/+Z). Sumun käyttäminen ei välttämättä ole hyvä idea iOS-laitteille suunnatussa pelissä, mutta jos sumua käytetään, olisi sen hyvä täsmätä skyboxin värien kanssa. Skyboxit tuottavat peliin kuusi piirtokomentoa, joten niiden käyttäminen iOS-laitteilla ei ole välttämättä järkevä ratkaisu. Suositeltavampaa on tehdä oma skydome.

### 3.11 NORMAALIKARTAT

Normaalikarttatekstuurien käyttäminen on mahdollista iPhone 3gs - ja sitä uudemmissa laitteilla. Tämä johtuu siitä, että kyseiset laitteet tukevat OpenGL ES 2.0 -standardia. OpenGL ES 2.0 -standardilla voidaan taas toteuttaa shadereita, jotka tukevat normal map -tekstuureja.

Normaalikarttatekstuurien käyttäminen iOS-laitteilla ei ole välttämättä hyvä idea, koska laitteiden grafiikkaprosessori ei ole riittävän nopea käsittelemään normaalikarttoja. Jos normaalikarttoja aiotaan käyttää, tulee sitä miettiä tarkoin etukäteen. Yksityiskohtien leipominen voi monessa tilanteessa olla järkevämpi vaihtoehto, sillä se on aina nopeampi tekniikka millä tahansa alustalla.



### 3.12 KORKEUSKARTAT

Unity-ohjelmassa on oma editori maastojen luomiseen, mutta maastoja voidaan luoda myös kuvankäsittelyohjelmalla tehdyillä korkeuskartoilla. Kuvien täytyy olla harmaasävykuvia ja RAW-tiedostomuodossa. Tiedostojen pitää käyttää täyttä 16-bitin resoluutiota. Myös oikean maailman maantiededataa voidaan käyttää Unity-ohjelmassa maaston luomiseen. Unity-ohjelmalla tehtyjä maastoja voidaan muokata millä tahansa korkeuskarttaeditorilla esimerkiksi Terragen-, Bryce- tai Photoshop-ohjelmalla. Tämä on mahdollista siksi, että Unity-ohjelmasta on mahdollista viedä korkeuskartat RAW-muotoon. Maastot saattavat aiheuttaa iOS-laitteilla suorituskyky ongelmia, joten niiden käyttämisessä täytyy olla hyvin tarkkana.

### 3.13 TEKSTUURIATLAKSET

Tekstuuriatlas on resoluutioltaan isokokoinen kuva, joka sisältää monen eri 3D-mallin tekstuurit. Kun 3D-mallit jakavat saman tekstuurin, on mahdollista yhdistää objektien piirtokomentoja. Myös tekstuuriatlakset täytyy olla tehty kahden potenssiin.

Vaikka tekstuurien yhdistäminen yhdeksi tekstuuriatlakseksi vaatiikin enemmän työtä, se on useimmiten kannattavaa. Kun useat objektit käyttävät samaa materiaalia, se maksaa ylimääräisen työn takaisin. Optimointi säästää myös prosessorin suorituskykyä.



Kuva 7. Tekstuuriatlas.

### 3.14 TEKSTUURIEN PAKKAAMINEN UNITY-OHJELMASSA

Ainoa tekstuurien pakkausmuoto, jota iOS-laitteet tukevat on PVRT. PVRT-pakkausmuodolla voidaan pakata tekstuureja, jotka ovat RGB- tai RGBA-väritilassa. Tekstuurien täytyy olla myös neliönmallisia, jotta ne voidaan pakata. PVRT-pakkausmuodolla voidaan pakata tekstuurit niin, että yksi pikseli vie kaksi tai neljä bittiä muistia. Pakkaamalla tekstuureita, voidaan oleellisesti vähentää muistin käyttöä sekä muistin kaistanleveyttä. Muistin kaistanleveydellä tarkoitetaan nopeutta, jolla tietoa voidaan lukea muistista ja se on yleensä hyvin rajallinen mobiililaitteissa. Pakatut tekstuurit käyttävät vain murto-osan muistin kaistanleveydestä mitä tarvittaisiin pakkaamattoman 32 bittisen RGBA tekstuurin käyttämiseen. Käytännössä tekstuurien pakkaaminen tarkoittaa nopeampia latausaikoja, pienempää muistinkäyttöä sekä suorituskyvyn merkittävää parantumista.

Jotkut läpinäkyvyyttä sisältävät PVRT-muotoon pakatut tekstuurit ovat alttiita näkyville artefakteille. Tällaisissa tapauksissa kannattaa PVRT-formaatin parametreja muokata suoraan kuvankäsittelyohjelmassa. Parametreja voidaan muokata, jos kuvankäsittelyohjelmaan on asetettu PVR vienti lisäosa tai käyttämällä PVRTexTool-työkalua. PVRTexTool-työkalun on tehnyt Imagination Tech, joka on myös PVRT-formaatin luoja. Työkalun voi ladata osoitteesta: <http://www.imgtec.com/powervr/insider/powervr-pvrtex tool.asp>.

Jos PVRT-muotoon pakatut tekstuurit eivät anna tarpeeksi hyvää kuvanlaatua tai haluat terävemmän kuvan, kannattaa harkita 16-bittisten tekstuurien käyttämistä RGBA-väritilassa olevien tekstuurien sijaan. Tämä voi olla järkevää esimerkiksi käyttöliittymän tekstuureissa. Tällä tavalla voidaan vähentää muistin kaistanleveyttä puolella.

### 3.15 VALAISU

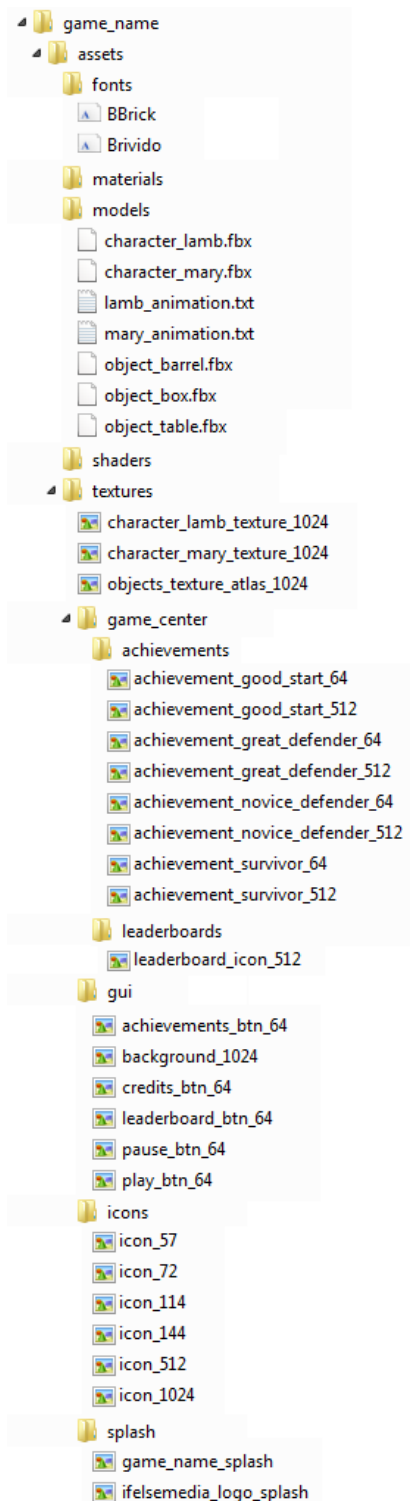
Dynaaminen valaisu per pikseli lisää merkittävästi jokaisen vaikutuksenalaisen pikselin renderöintikustannuksia. Se voi johtaa siihen, että objektit renderöidään useaan kertaan. Sellaisia tilanteita, joissa pikselivaloja olisi useampia objektin valaisemiseksi, tulisi välttää. Suuntavaloja tulisi käyttää niin pitkään kuin on mahdollista.

Dynaaminen valaisu per verteksi voi lisätä merkittävästi verteksien muunnoskustannuksia. Myös tällaisissa tilanteissa pitää yrittää välttää, ettei objektia valaise useat valot. Valaisun leipominen on paljon tehokkaampaa staattisissa objekteissa.

Näkymän staattisten objektien valaisu kannattaa leipoa tekstuureihin käyttämällä Unity-ohjelman sisäänrakennettua valokartta-työkalua (*Lighmapper*). Prosessi, jossa generoidaan valokartoitettu ympäristö, vie hieman enemmän aikaa kuin valojen asettaminen näkymälle. Sen hyödyt ovat silti selkeät, sillä se toimii paljon nopeammin ja näyttää paljon paremmalta, koska globaali valaisu voidaan leipoa ja valokartta-työkalu voi tasoittaa aikaansaattua tulosta.

## 4 KANSIORAKENNE JA NIMEÄMISKÄYTÄNTÖ

Useat eri henkilöt saattavat joutua projektissa käyttämään tekemiäsi tiedostoja. Tämän vuoksi tiedostojen nimien pitää kuvata mahdollisimman kattavasti tiedoston sisältöä. Ifelse Media Oy käyttää tiedostojen ja kansioden nimeämisessä tapaa, jossa välilyönti korvataan alaviivalla. Tiedostot ja kansiot pitää nimetä mahdollisimman kuvaavasti. Esimerkki tiedoston nimeämisestä: texture\_mary\_1024.



Peliprojektin kansiorakenne tulee olla kuvan viisi mukainen grafiikan osalta. Kansiorakenne on suunniteltu niin, että se on mahdollista siirtää suoraan Unity-ohjelman kansiorakenteeseen. Jos et tarvitse jotain kuvassa esiintyviä kansioita, ei niitä tarvitse kansiorakenteeseen lisätä. Esimerkiksi game\_center-kansiota ei välttämättä tarvita kaikissa projekteissa.

Kuva 8. Esimerkki kansiorakenteesta.

## 5 SOVELLUSKUVAKE

Sovelluskuvake on kuva, joka tulee käyttäjän iOS-laitteen koti-valikkoon sovelluksen lataamisen jälkeen. Sovelluskuvaketta koskettamalla avataan itse sovellus. Jokainen sovellus tarvitsee sovelluskuvakkeen. Jos sovellus on peli, käytetään sen sovelluskuvatta myös Applen Game Center -palvelussa.

Kun iOS-laitteiden käyttöliittymä näyttää sovelluksen kuvakkeen laitteen koti-valikossa, se automaattisesti lisää kuvakkeeseen seuraavat visuaaliset efektit: pyristetyt reunat, varjostuksen ja heijastuksen. Heijastus on mahdollista poistaa käytöstä, kun sovellus laitetaan App Store -kauppapaikkaan.

Sovelluksen kuvake pitää tallentaa PNG-formaattiin. Kuvan väripalettia ei tarvitse rajoittaa Web-turvallisiin väreihin. Sovelluksen kuvakkeen pitää täyttää seuraavat kriteerit ennen kuin se lähetetään App Store -kauppapaikkaan: Kuvassa pitää olla 90 asteen kulmat, kuva ei saa sisältää pudotusvarjoa, kuvassa ei saa käyttää läpinäkyvyyttä eikä kuvassa saa olla heijastusta, jos siinä aiotaan käyttää iOS:n tarjoamaa heijastusta.

Esimerkkikuva kuvakkeesta. Kuvassa esitettyjen efektien tarkoituksena on havainnollistaa iOS:n automaattisesti lisäämiä efektejä.



Kuvake ennen iOS:n lisäämiä efektejä.



Kuvake, johon iOS on lisännyt pyöristetyt kulmat, varjostuksen ja heijastuksen.



Kuvake, johon iOS on lisännyt pyöristetyt kulmat ja varjostuksen. Automaattinen heijastus on otettu pois käytöstä.



Kuvake, johon on tehty reunukset ennen iOS:n lisäämiä efektejä.



Kuvake, johon tehty reunat ja iOS on lisännyt pyöristetyt kulmat ja varjostuksen. Automaattinen heijastus on otettu pois.

Kuva 9. Kuvakkeen käyttäytyminen iOS-laitteella.

## Sovelluksen kuvakekoot

Laite	Pikselikoko
iPhone, iPod touch	57x57
iPad	72x72
Korkearesoluutio iPhone	114x114
Korkearesoluutio iPad	144x144

## Sovelluksen kuvakkeen koot App Store -kauppapaikkaan

Laite	Pikselikoko
iPhone, iPad	512x512
Korkearesoluutio iPhone, iPad	1024x1024

## Sovelluksen kuvakkeen kulmanpyöreys

Pikseli koko	Kulmanpyöreys pikseleinä
57x57	10
72x72	12
114x144	24
512x512	90
1024x1024	180

## 5.1 GAME CENTER

Game Center on Applen palvelu, jolla voidaan luoda peliin achievementteja ja leaderboardeja. Game Center -palvelun avulla voidaan peliin luoda myös moninpelimahdollisuus. Jos Game Center -palvelun avulla luodaan peliin achievementteja tai leaderboardeja, täytyy palveluun lähettää kuvia. Kuvat voivat olla .jpeg-, .jpg-, .tif-, .tiff- tai .png-muodossa. Kuvien resoluutio pitää olla 512x512, vähintään 72 dpi ja niiden täytyy käyttää RGB-väritilaa. Läpinäkyvyyttä ei suositella kuvissa käytettäväksi.

## 6 FONTIT

Fontteja voidaan käyttää GUI tekstinä tai Text Mesh -komponentissa. Text Mesh -komponentti generoi 3D-geometrian, jossa näkyy kirjoitettu teksti ja asettaa tekstin 3D-näkymään. Sitä voidaan käyttää esimerkiksi opaskylttien tekemiseen. GUI Text -komponenttia kannattaa käyttää, kun halutaan tehdä yleistä kaksiulotteista tekstiä käyttöliittymiin. Fonttien, joita aiotaan käyttää sovelluksessa, pitää olla .ttf-tiedostoja.

## 7 3D-MALLIEN TYÖVÄLINEET JA TALLENNUSMUODOT

Unity-ohjelma tukee useita eri 3D-ohjelmia ja niiden erilaisia tallennusformaatteja. Ohjelma tukee seuraavia yleisiä 3D-tallennusmuotoja: .fbx, .dxf, .obj, .dae ja .3DS. Seuraava taulukko kuvaa Unity-ohjelman tukemia 3D-ohjelmia sekä ohjelmien tuettuja tallennusmuotoja.

Ohjelman nimi	Tuetut tiedostomuodot
Cinema 4D	.c4d
Blender	.blend
Maya	.mb, .ma
3ds Max	.max
Cheetah3D	.jas
Modo	.lxo
Lightwave	.fbx

Ifelse Media Oy käyttää 3D-mallien tuotannossa Cinema 4D R12 Prime -ohjelmaa ja 3D-mallien tallennusmuotona FBX-formaattia. Mallinnusohjelmasta riippuen mobiilipelien 3D-mallit on toimitettava FBX-muodossa. Lisätietoa muiden 3D-mallinohjelmien vientiasetuksista löytyy osoitteesta:

<http://unity3d.com/support/documentation/Manual/HOWTO-importObject.html>

## 8 MUISTILISTA MOBIILIPELIGRAFIIKAN OPTIMOINTIIN

- Verteksien määrä on syytä pitää alhaisena.
- Alle 40 000 verteksiä per kuvaruutu, kun tähdätään iPhone 3gs - ja uudemmille laitteille.
- Jos käytät Unity-ohjelman valmiita shadereita, käytä mieluiten Mobile-kategoriasta. Mobile/VertexLit on tällä hetkellä nopein shaderi.
- Pidä materiaalien määrä per scene alhaisena - Käytä samoja materiaaleja objektien kanssa niin paljon kuin pystyt.
- Käytä tekstuurien pakkaamisessa PVRT-formaattia aina, kun se on mahdollista. Muissa tapauksissa kannattaa käyttää 16-bittistä tekstuuria 32-bittisen sijaan.
- Älä käytä useita pikselivaloja (*Pixel Lights*) ellei ole aivan pakko. Kannattaa valita mieluummin vain yksi pikselivalo geometrian valaisemiseen.
- Älä käytä dynaamisia valoja ellei ole aivan välttämätöntä. Leivo mieluummin valaistus.
- Käytä vähemmän tekstuureja per kappale.
- Vältä alfa-taustausta, valitse mieluummin alfa-sekoitus.
- Älä käytä sumua ellei ole aivan välttämätöntä.